Investigation of Methods for Increasing the Reliability of Highly Integrated Non-volatile Memories on System Level

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus-Senftenberg

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

genehmigte Dissertation

vorgelegt von

Magister Ingenieur
Patryk Skoncej
geboren am 19. Oktober 1986 in Stargard Szczeciński, Polen

Gutachter: Prof. Dr.-Ing. Rolf Kraemer
Gutachter: Prof. Dr.-Ing. Heinrich Theodor Vierhaus
Gutachter: Prof. Dr. Michael Gössel

Tag der mündlichen Prüfung: 9. Dezember 2014

Abstract

Conventional semiconductor memories are facing many challenges concerning their yield, reliability, testability, and manufacturability as the feature size decreases. Although they are used in the vast majority of electronic devices, their applicability for upcoming digital systems is questionable. On the other hand, due to unprecedented development of mobile devices even faster, denser, and more power-efficient semiconductor memories are required. As a consequence, many researchers and system designers are seeking new memory solutions. The greatest attention is paid to solid-state, non-volatile memories (NVMs) such as PCRAMs, MRAMs/STT-MRAMs, FeRAMs, and RRAMs. Due to their promising features like non-volatility, low-power consumption, and great scalability they are expected to meet the challenging demands of future digital systems.

Unfortunately, despite all advantages they offer, emerging NVMs pose some peculiar characteristics like limited endurance, variable data retention time, or vulnerability to external factors. On top of that, they are still in early-maturity state where their fabrication processes are not of high quality and are prone to high variations. Because of that, emerging NVMs may suffer from permanent faults which can occur right after production or in the field, during their operational time. As a consequence, the reliability of new memory technologies requires special management and great improvement.

The thesis introduces system-level approach aimed at comprehensive reliability management of existing and emerging NVMs. It presents novel on-line repair techniques which focus on specific issues of NVMs. The block-level repair manages post-production faults in the memory array. The word-level repair aims at hard faults caused by wear-out memory cells. Finally, the error-correcting code with increased hard-error correction capability handles soft and hard errors in the memory array.

Because proposed techniques are based on similar principles, they can be combined into a consistent system. Depending on the way how they are connected, different repair schemes can be achieved. Moreover, by merging them into the system a synergistic effect can be produced where the achieved memory reliability improvement is greater than the sum of reliability improvements achieved with their standalone implementations.

Further in the thesis, such a consistent repair system is presented. Next, its effectiveness, repair capabilities, and applicability for an embedded system are evaluated. In addition, the achieved synergistic effect is described and quantified.

Kurzfassung

Konventionelle Halbleiterspeicher stehen vor vielen Herausforderungen bezüglich. ihrer Fertigungsausbeute, Zuverlässigkeit, Testbarkeit sowie der Herstellbarkeit bei immer weiter sinkenden Strukturgrößen. Obwohl sie in der überwiegenden Mehrzahl elektronischer Geräte verwendet werden, ist ihre Anwendbarkeit für zukünftige digitale Systeme fraglich. Auf der anderen Seite sind durch die beispiellose Verbreitung mobiler elektronischer Geräte sogar schnellere und energieeffizientere Halbleiterspeicher mit wachsenden Speicherkapazitäten erforderlich. Als Konsequenz suchen viele Forscher und Systementwickler nach neuen Speicherlösungen. Das größte Augenmerk liegt dabei auf integrierten, nichtflüchtigen Speichern (NVMs) wie PCRAMs, MRAMs/STTMRAMs, FeRAMs und RRAMs. Aufgrund ihrer vielversprechenden Eigenschaften wie Nichtflüchtigkeit, niedriger Energieverbrauch und gute Skalierbarkeit, könnten sie die anspruchsvollen Anforderungen an zukünftige digitale Systeme erfüllen.

Leider haben diese neuen NVMs, trotz aller Vorteile, die sie bieten, auch einige unerwünschte Eigenschaften wie eine begrenzte Lebensdauer und Datenhaltung, sowie eine besondere Anfälligkeit für externe Störungen. Obendrein befinden sie sich in einem frühen Reifestatus, das heißt, sie werden mit Fabrikationsprozessen geringerer Qualität hegestellt und sind somit anfälliger bei Prozessabweichungen. Aus diesem Grund leiden die neuen NVMs an Speicherfehlern, die schon direkt nach der Produktion oder später während ihrer Nutzung auftreten. Daher ist für eine hohe Zuverlässigkeit der neuen Speichertechnologien ein spezielles Speichermanagement erforderlich.

Die vorliegende Dissertation liefert einen Ansatz für ein umfassendes Zuverlässigkeitsmanagement sowohl bereits existierender als auch neuer NVMs auf Systemebene. Sie präsentiert neuartige Online-Reparaturtechniken mit dem Schwerpunkt auf spezifische Eigenheiten der NVMs. Die Block-Level-Reparatur korrigiert permanente Fehler des Speicherarrays, die bereits nach der Produktion auftreten. Die Wort-Level-Reparatur dagegen soll schwere Fehler korrigieren, die durch den langsamen Verschleiß von Speicherzellen verursacht werden. Schließlich kann auch ein erweiterter Fehlerkorrektur-Code verwendet werden, um sowohl kurzzeitig auftretende Bitfehler als auch permanente Fehler im Speicherarray zu korrigieren. Weil die hier vorgeschlagenen Techniken auf ähnlichen Prinzipien beruhen, sind sie in einem konsistenten System kombinierbar. Abhängig von der Art und Weise wie sie kombiniert werden, können verschiedene Reparatursysteme entstehen. Darüber hinaus kann durch Kombination der Systeme eine Synergiewirkung erreicht werden, bei der die erzielte Verbesserung der Zuverlässigkeit größer ist als die Summe der Verbesserungen, die mit den einzelnen Implementierungen erreicht werden können.

In der vorliegenden Dissertation wird solch ein konsistentes Reparatursystem vorgestellt. Weiterhin werden die Effektivität, die Reparaturfähigkeit und die Eignung für eingebettete Systeme ausgewertet. Schließlich wird die erzielte Synergiewirkung beschrieben und quantifiziert.

I dedicate this work to my other halves,
my beloved wife Aleksandra and my dear daughters, Antonina and Zofia.

Contents

List of Figures

List of Tables

Research context and motivation

The amount of information being transferred, processed, and stored is continuously increasing at unprecedented rate. Our need for having constant access to the information which is important for us has driven the digital revolution. Because of the technology advancements, year after year we are able to develop smaller, faster, and more power-efficient electronic devices which fulfill our urge for being *always connected*. In order to preserve this trend much research and development is needed in the areas of new materials, design concepts, and fabrication processes. Especially, the dependability of semiconductor devices is of utmost concern as it decreases with smaller technology nodes.

In order to keep-up with the amount of information required to fulfill our needs, electronic devices are being equipped with larger and faster semiconductor memories. Unfortunately, commonly used semiconductor memories face reliability and manufacturability concerns as the feature size decreases. Because of that, new memory technologies are being developed. The biggest focus is put on non-volatile semiconductor memories as they are foreseen as suitable memories for power-constrained mobile devices. Although today, novel NVMs are being produced and shipped, they are still in early-mature phase where further improvement concerning they fabrication, scalability, and reliability is required. Especially, characteristics related to their reliability require special attention and management.

Most of existing NVM management techniques were derived from techniques developed for conventional semiconductor memories. Because of that, they often do not provide an optimal management. On the other hand, techniques which specifically address novel NVMs:

- are complex,
- require user/operating system attention,
- require changes in the internal components of the memory, and/or
- are aimed mostly at large digital systems.

Since the biggest advantage of non-volatile memories is that they preserve stored data even when the power is lost, they are well suited as embedded memories for mobile devices (e.g, sensor nodes, mobile phones).

Another aspect related to NVMs which has to be addressed is their usability and applicability in digital systems. Because NVMs possess peculiar characteristics, their implementation and management in the system is often cumbersome. To reduce the

design time and to decrease the gap between *what is done* with emerging technologies and *what can be done*, novel NVM management techniques should be developed.

As a consequence, management techniques aimed at NVMs should be simple, transparent for the user, and have minimal impact on other components in the system.

Research contributions and list of own publications

The research conducted in the course of the PhD project was driven by the need to facilitate the reliability management of existing and emerging NVMs in digital systems. Most of research outcomes were documented, submitted to peer-review conferences, and published in conference proceedings. Further, the most important scientific publications are summarized.

Non-volatile Memory Controller Design using Fault-tolerant Techniques for Memory Reliability Improvement
Presented at Smart Systems Integration (SSI 2011), March, 2011

In [92] a first attempt to provide a comprehensive management for the non-volatile memory is presented. The paper describes a memory controller aimed at improving the reliability of the NOR flash memory. The proposed controller implements an error-correcting code (ECC) based on the Hsiao code for each memory word. In addition it provides a block-based redundancy repair with configurable memory block size.

Getting Ready for Non-volatile Memories
Presented at 2nd Workshop on Resilient Architectures, December, 2011

In [91] a review of the state-of-the-art NVM management techniques is provided. The paper outlines challenges related with emerging NVMs and summarizes advantages and disadvantages of existing solutions aimed at overcoming these challenges. Moreover, it discuss areas on which future management solutions should focus.

Single Error plus Single Erasure Correction with Redundancy Repair Scheme for Memory Reliability Improvement
Presented at Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ 2012), February, 2012

In [89] a modified ECC design able to correct double-bit error is presented. In 1979 Walker et al. [98] proposed a decoding algorithm together with a decoder architecture capable of correcting one erasure and one single-bit error. The paper presents an improvement of the method proposed in [98] able to reduce required area and facilitate memory system usage. Further in the paper the applicability of the modified ECC design for existing and emerging NVMs is discussed.

WBR - Word and Block-Level Hard Error Repair for Memories

Presented at Non-Volatile Memory Technology Symposium (NVMTS 2012), October, 2012

In [90] a novel memory repair approach aimed at emerging NVMs is presented. The repair mechanism called WBR is based on replacing defective data blocks with spare ones for every memory block separately. Repair procedures implemented in the WBR can be applied on memory word- and block-levels offering different repair speeds and correction capabilities. Further in the paper the WBR is compared against state-of-the-art repair solutions. According to comparison results the WBR provides better memory lifetime improvement for small memory word sizes and achieves comparable results for wider memory words. In contrast to the state-of-the-art techniques, WBR can be applied to word sizes as small as 16 bits imposing less than 12.5% of additional bit overhead. The WBR can be implemented purely in hardware, e.g. in form of a memory controller and is complementary to existing wear-leveling techniques.

Fault Injection Framework for Embedded Memories

Presented at IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2013), October, 2013

In [87] a fault injection framework for semiconductor memories is proposed. In principle, the fault injection framework provides a memory model based on expected memory endurance and usage scheme. The generated model can be further used in system simulations and emulations. The main purpose of the framework is to facilitate the evaluation of reliability-improving techniques and applicability of emerging NVMs for specific systems. Further in the paper the correctness of the framework is evaluated and its imposed overheads are reported.

Low-Overhead Fault Injection Simulation and Emulation Technique for Embedded Memories

Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ 2014), February, 2014

In [88] new results regarding the fault injection framework are presented. The paper reports results from the framework's evaluation concerning its impact on system's performance. As presented in the paper, the performance of the example system equipped with generated memory model was lower than performance of the reference system by 1%.

Structure of the thesis

Chapter 1 consists of two parts. The first part provides basic information about existing and emerging semiconductor memories with the emphasis on emerging non-volatile memories. It presents their working principles and outlines challenges related to their reliability. The second part focuses on reliability management techniques developed for semiconductor memories in general. Further, it provides in-depth overview of commonly-used and state-of-the-art fault-tolerance solutions for non-volatile memories.

Chapter 2 presents novel memory repair techniques aimed at existing and emerging non-volatile memories. It provides thorough analysis of their repair mechanisms and discuss their applicability for non-volatile memories. Next, it shows preliminary simulation results related to their repair effectiveness. Finally, the second chapter presents a combination of proposed techniques aimed at comprehensive non-volatile memory management.

Chapter 3 focuses on the evaluation of the comprehensive approach presented in the second chapter. It consists of three parts. In the first part, the framework used for the evaluation is described and its correctness is justified. The second part presents simulation procedure together with simulation results regarding the comprehensive approach. The third part of the chapter describes embedded system where the comprehensive approach was implemented. Further, it presents results from system emulations performed in the FPGA.

Chapter 4 concludes the thesis. It summarizes previous chapters and provides information regarding further investigations and research.

Chapter 1

Introduction

1.1 Semiconductor memories

A semiconductor memory is an electronic data storage device implemented on a semiconductor integrated circuit (IC). It has the ability to store and process data in digital form and has no moving, mechanical parts. Semiconductor memories are used to store user data, program code, and other information which can be referenced in desired time by a processing unit (e.g. microprocessor, microcontroller).

A semiconductor memory has to provide the ability to store data (memory write operation) at desired memory location defined by memory address. Further, data stored in the memory must be retrievable (memory read operation) from the same location in unchanged form. Moreover, semiconductor memories have to have the property of random access, i.e., data stored at different memory locations should have the same/similar access time.

Generally, data stored in a semiconductor memory has the form of memory words. Each memory word is typically a set of 2^n bits (e.g. 8, 16, 32, etc.) and has own, distinctive address by which it can be referenced.

The core of a semiconductor memory consists of a memory array. The memory array is a collection of storage elements called *cells* which are connected to word and bit lines (Fig. 1.1) [56]. Based on the technology and storage mechanism used to implement storage elements, a single memory cell can store single or multiple bits. A memory cell which can store a single bit is referred as single-level cell (SLC), whereas cell which can store multiple bits is referred as multi-level cell (MLC).

The memory array is surrounded by row and column decoders, column input/output circuitry, and control and interface logic (not depicted in Fig. 1.1) [39], [56], [48]. Address row decoder is used for selecting a single word line from the memory array. Address column decoder is used for selecting multiple bit lines from the memory array. Column input/output circuitry is responsible for managing data storage and retrieval to/from memory cells and consists mainly of write drivers and read sense amplifiers. Control logic is responsible for managing memory internal modules whereas interface logic is responsible for proper communication between the memory and the external world.

In a typical semiconductor memory, during memory read/write operations n address signals are decoded by the row decoder to select a single word line from the memory array (Fig. 1.1). The rest of address signals (m) is decoded by the column decoder to select i bit lines. Next, i memory cells located on the intersections of selected word line

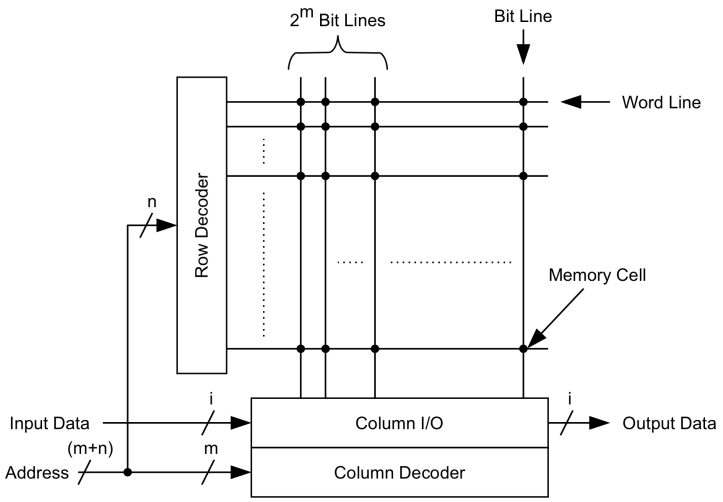


Figure 1.1: Basic memory structure.

and bit lines are processed by i read sense amplifiers/write drivers. The number of read sense amplifiers/write drivers defines the number of memory data output/input signals.

The number of memory cells and the way in which they are arranged in the memory array have a significant impact on the memory access time and power consumption. That is, the longer bit and word lines are, the bigger their capacitance resulting in increased delay and power dissipation. Because of that, the main part of memory design process is focused on uniform distribution of bit- and word-line lengths and proper arrangement of memory cells. Moreover, since the number of bit lines is usually larger than the number of required memory data input/output signals, multiple bit lines often share a single read sense amplifier and write driver. The memory array is divided into column blocks, each consisting of 2^m bit lines (Fig. 1.1). In this memory design, during read/write operations m address signals are used to select one of 2^m bit lines from each column block which are further processed by column input/output circuitry.

Depending on the location of the memory in respect to the processing unit, two types of the semiconductor memories can be distinguished: standalone and embedded [18]. Standalone memories, in contrast to embedded counterparts, are devices which are not directly built into the same chip where a processing unit resides. Moreover, standalone memories can achieve greater densities as they can benefit from memory optimized fabrication process. They are mass produced devices which typically have serial[1] interface in order to reduce the number of chip-to-chip connections. On the contrary, embedded memories can be specifically designed in order to fulfill required system specifications.

[1]Serial memory interface provides sequential, bit by bit reading/writing from/to memory typically over a single data output/input line.

Although they are usually more complex to fabricate, they can achieve higher bandwidth with wide parallel[2] interfaces. Furthermore, because of short connection wires and small on-chip input/output drivers, significant power and delay reductions can be achieved with embedded memories.

Generally, semiconductor memories are classified in respect to the functionality they fulfill in a digital system. As a result, two main categories of semiconductor memories can be distinguished: random access memories (RAMs) and read only memories (ROMs) (Fig. 1.2).

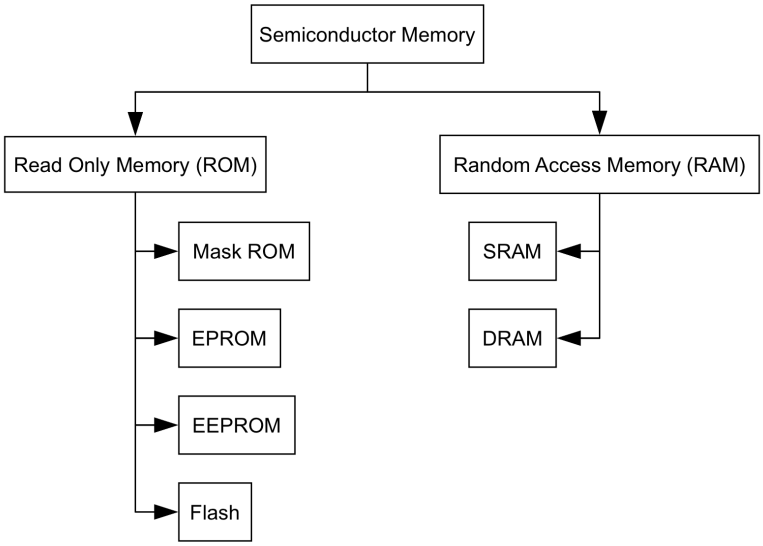


Figure 1.2: Semiconductor memory classification.

Read only memories are memories whose content, once programmed, is hard or even impossible to change. Moreover, even if content can be changed, significantly more effort (e.g. time, energy) is required to update stored data than to read data. The most common feature of ROMs is that they are non-volatile. Data stored in the memory remains present even if the power connected to the memory is lost. This feature is the main reason why they are typically used for storing e.g., boot codes, look-up tables, and/or parameters for digital signal processors. Although ROMs and RAMs are classified as separate groups, ROM memories also have the property of random access. Conventional[3] representatives of ROMs are Mask ROM and memories based on floating-gate transistors (EPROM, EEPROM, and flash).

Random access memories are memories which offer very fast times for data storage and retrieval. They can serve virtually unlimited (in respect to the system lifetime) number of write/read operations and are typically used as caches, main memories, and/or data

[2]Parallel memory interface provides simultaneous access to several data input/output signals. Typically, in parallel memory interface the data bus to/from memory has the width of a memory word.

[3]The term *conventional memories* refers to mature memory technologies. That is, memory technologies which are well-known and has been used for a long time in digital systems.

buffers. In comparison to ROMs, majority of RAMs are volatile. Once the power connected to the memory is switched off, data stored in the memory is also lost. Conventional representatives of RAMs are SRAM and DRAM.

Since the introduction of microcontrollers in the seventies, the importance of semiconductor memories is constantly rising. Nowadays, semiconductor memories play an essential role in almost every modern electronic system. Moreover, since the development of system-on-chip (SoC) design concept their impact on digital system increases. That is, system's performance, power consumption, yield, and reliability strongly depend on the memory embedded into the system. In present times, the area occupied by embedded memories can even dominate the area of SoC (Fig. 1.3) [24].

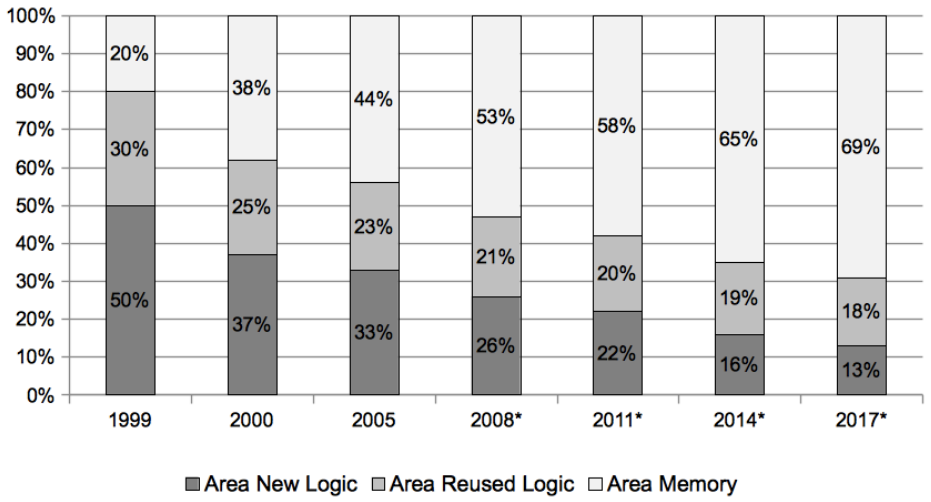


Figure 1.3: Roadmap for die area partitioning 1999-2017. Figure adapted by author from [24].

Due to fast development of mobile devices the complexity of today's SoCs is getting higher. As more and more functions are implemented into a single chip, denser, faster, cheaper, and less power-hungry embedded memories are required. Although conventional semiconductor memories are still fulfilling designers expectations, their existence at smaller feature sizes is questionable. As a result, many researchers and system designers are looking for new memory technologies. The best solution would be to have a semiconductor memory which would possess the characteristics of RAMs and non-volatility of ROMs. This hypothetical memory is often referred in research community as *universal memory*. The universal memory is expected to provide data non-volatility, fast read and write access times, very good reliability and endurance[4], low-power consumption, great scalability, and very good integration with the CMOS technology. Another benefit of

[4]The memory endurance is defined as the number of memory read/write operations which can be reliably performed on the memory. Usually, the information about memory endurance is provided by the memory manufacturer and reflects the minimum endurance of a memory cell.

possessing a universal memory is that existing memory hierarchy could have been simplified and modified. Presently, memory hierarchy consists of several levels of storage devices which have different storage capabilities and access times. As a consequence, the system's performance is often not optimal. With universal memory, different memory technologies which are currently implemented in digital systems could have been replaced with a universal one.

As mentioned earlier, conventional memories cause manufacturing and implementation challenges as the feature size decreases. Moreover, because of their characteristics, they do not fit very well into a universal memory concept. As a result, new memory technologies are being developed. According to the International Technology Roadmap for Semiconductors (ITRS) [82], the greatest interest and research focus is placed on phase-change RAMs (PCRAMs), magnetoresistive RAMs (MRAMs), and ferroelectric RAMs (FeRAMs). The ITRS mentions also resistive RAMs (RRAMs) as potential solution for future memory systems. While PCRAMs, MRAMs, and FeRAMs are currently in volume production, RRAMs still require a lot of research and development effort to demonstrate their viability.

Basic information about conventional memories, their working mechanism and potential challenges are further presented in Section 1.1.1. Emerging memory technologies (PCRAMs, MRAMs, and FeRAMs) are described in Section 1.1.2.

1.1.1 Conventional memories

Static RAM

Static random access memory (SRAM) is a volatile semiconductor memory which offers very fast read/write accesses, low-power consumption, and has virtually unlimited endurance. The term static means that data stored in the memory can self-maintain as long as power supply is provided [56].

The most common implementation of SRAM memory cell is called *6T* cell. The cell structure is depicted in Figure 1.4. The storage element consists of two, cross-coupled inverters ($M1$, $M2$ and $M3$, $M4$) which form a feedback loop [56]. The access to the storage element is granted through two N-channel metal-oxide-semiconductor (NMOS) field-effect transistors ($M5$ and $M6$) which act as selecting switches. The memory cell is connected to the word line (WL) and two bit lines (BL and $\overline{BL}$).

When WL is connected to the ground, $M5$ and $M6$ transistors are switched off, and power supply V_{DD} is provided to the storage element, the memory cell is in a standby mode. In the standby mode the cross-coupled inverters reinforce themselves and the binary data Q and $\bar{Q}$ remain at their original values [56].

In order to read/write from/to SRAM cell, the cell has to be selected, i.e., two access transistors $M5$ and $M6$ have to be switched on by connecting WL to V_{DD}. During read operation, two bit lines (BL and $\overline{BL}$) are connected to the sense amplifier which recognizes logic state stored in the memory cell, either '0' or '1'. Further, the logic state is transferred to the memory output.

During write operation, logic data which comes from the input pad is transferred to

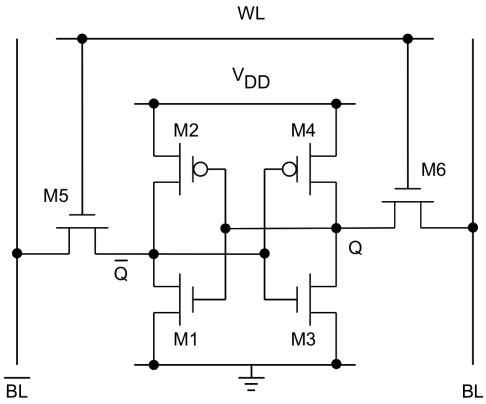


Figure 1.4: SRAM 6T cell structure.

the write circuitry. Next, the write circuitry sets bit lines to appropriate voltage levels. Because the write drivers are stronger than the transistors in cell inverters, the input data is forced onto the cell.

Since SRAM cell can be implemented with standard transistors, the SRAM technology offers great compatibility with CMOS technology. Moreover, although the memory cell is larger and more expensive to fabricate than e.g., DRAM cell, SRAM offers faster read/write accesses and lower power consumption. Because of that, SRAMs are very good solutions for on-chip caches, buffers, or small main memories for microcontrollers.

Unfortunately, the fabrication process of transistors is becoming a challenge as the feature size decreases. Process variations occurring during the fabrication phase cause higher defect densities, transistor parameters variations, and new reliability failure mechanisms [52]. The problem is more severe when it comes to the SRAM fabrication. Since the memory design process is set to achieve large bit densities, low-power consumption, and good performance, SRAM transistors are thus more vulnerable to failures. The failure mechanisms caused by negative bias temperature instability (NBTI), time dependent dielectric breakdown (TDDB), and hot carrier degradation (HCI) can influence cell's ability to reliably preserve stored data [11]. Moreover, as the technology node scales down, transistors are more susceptible to soft errors[5] [93]. That is, the amount of charge required to influence circuit's output or state is getting smaller and smaller. As a result, it will become increasingly difficult to meet required reliability levels for SRAM in upcoming digital systems without improving fabrication process or developing new fault protection/management mechanisms.

[5]Soft errors are unwanted, not permanent changes in the output or state of a circuit. They are caused by charges generated by an energetic particles due to direct or indirect ionization causing voltage and current swings in the circuit. The main sources of energetic particles are alpha particles (main source: fabrication process chemicals, packaging materials) and neutrons (main source: cosmic rays).

Dynamic RAM

Dynamic random access memory (DRAM) is a volatile memory storage device which is mostly used as the main memory in personal computers and mobile devices. The DRAM memory cell is composed of a single capacitor and a single access transistor (Fig. 1.5) [56]. Presence or absence of a charge on the capacitor is used as storage mechanism. Because the capacitor tends to leak away, it is required to periodically refresh the storage element in order to preserve stored data. The requirement for periodic refresh is the main reason why the memory is called *dynamic* [56].

The DRAM cell structure is depicted in Figure 1.5. The memory cell is connected to the word line and bit line. In order to store data in a memory cell, first, the bit line has to be set to high or low voltage. Next, by switching on the selecting transistor, the capacitor in the selected cell is charged or discharged. Reading data from a DRAM cell requires pre-charging bit line to $V_{DD}/2$ and switching on the access transistor by connecting the word line to V_{DD} [56]. Next, depending on the presence or absence of a charge on a cell capacitor the voltage on the bit line changes. This change is then sensed by a sense amplifier and appropriate logic value '0' or '1' is driven to the memory output. During memory read procedure, the capacitance of a bit line has influence on the charge on the cell capacitor. Because of that, the read data has to be written back to the cell in order to restore the original capacitor state.

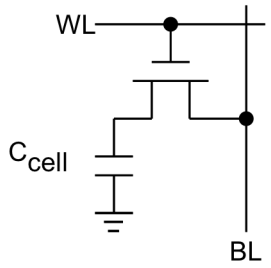


Figure 1.5: DRAM cell structure.

Because DRAM cell is smaller than an SRAM cell, the DRAM can achieve higher data densities. Moreover, since more data can be stored on the same chip area than in SRAMs, DRAMs are cheaper to fabricate. As for the drawbacks, DRAM is usually slower and requires periodic refresh of each memory cell in order to preserve data. As a consequence, DRAMs are more power hungry in comparison to their SRAM counterparts.

As earlier mentioned, SRAM storage mechanism is based on changing the state of cross-coupled inverters. In DRAM, on the other hand, the storage mechanism is based on charging and discharging cell's capacitor. Because the amount of the charge stored on the capacitor can be measured, there is a possibility to store more than one bit in a single DRAM cell. With MLC DRAM cell even higher data densities can be achieved for the cost of more complex design and fabrication processes [69], [58].

DRAM technology is facing many challenges with decreasing feature size. The main issues are related with capacitor scaling. As the memory cell size decreases, it is more difficult to maintain the volume of the capacitor so that required charge level remains constant [63], [101]. Moreover, smaller access transistors tend to have higher leakage current and have more problems with driving a sufficient amount of charge to the capacitors. Because of that, the data retention time[6] is getting shorter causing the need for more frequent refresh periods. Furthermore, the data retention time is also temperature dependent. At elevated temperatures, retention time of DRAM cells decreases exponentially [59].

Unlike in SRAMs, soft error rates in DRAMs are trending downwards and are several orders of magnitude lower [93]. This is caused by the fact that even with technology scaling, the charge level of cell's capacitor remains constant. Nevertheless, particle strike on the storage or select transistor can discharge memory cell causing data loss.

Mask ROM

Mask read-only memory (MROM) is a non-volatile memory which is the most basic and simplest memory structure which can be implemented in an integrated circuit. The storage element can be implemented by using only one transistor, thus high data densities and low-cost fabrication can be achieved. The implementation of logic '0' or '1' can be realized through the absence or presence of an NMOS transistor at the intersection of the word line and bit line (Fig. 1.6) [18].

Based on a way how the memory cells are arranged in memory array two ROM configurations exist: NOR, and NAND [18]. In NOR ROM cells are connected in parallel to the bit lines (Fig. 1.6a) which resembles the parallel connection of transistors in a CMOS NOR gate. In NAND ROM cells are connected in series (Fig. 1.6b) which resembles transistors connection in a CMOS NAND gate.

In the NOR ROM configuration, all bit lines are pulled to V_{DD}. Because of that, default output from the each bit line is '1' unless a switched-on transistor connected to the appropriate bit line will not short it to the ground. For example (Fig. 1.6a), if $WL0$ is switched to V_{DD} and the rest of word lines are shorted to the ground, the output of $BL0$ will be '1' because there is no switched-on transistor connected to the $BL0$ which can short it to the ground. In case of $BL2$, the transistor connected to the $WL0$ is switched on producing '0' at the output of $BL2$.

In the NAND ROM configuration, similarly like in NOR ROM, all bit lines are pulled to V_{DD}. Further, transistors are connected in series and not in parallel like in NOR ROM. As a result, in order to determine presence/absence of transistors connected to the selected word line, the selected word line has to be grounded and all other word lines have to be connected to V_{DD}. For example (Fig. 1.6b), if one would like to determine the absence/presence of transistors connected to the $WL0$, the $WL0$ has to be grounded and $WL1$ and $WL2$ have to be connected to V_{DD}. The output of $BL0$ will be '0' because switched-on transistor connected to the $BL0$ and $WL2$ will short $BL0$ to the ground.

[6]The data retention time is defined as an amount of time a memory cell can reliably preserve its state.

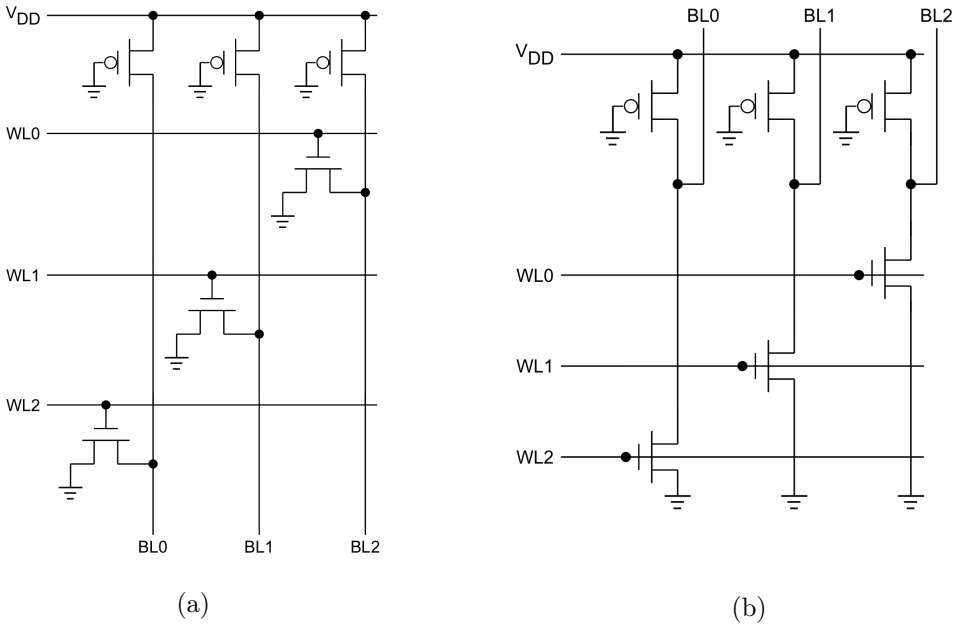


Figure 1.6: Different configurations of ROM array: NOR configuration (a), NAND configuration (b).

The output of $BL1$ will also be '0' because of the switched-on transistor connected to the $BL1$ and $WL1$. The output of $BL2$ will be '1' because the transistor connected to the $BL2$ and $WL0$ is not switched on.

The main benefit of using NAND structure over NOR is the size reduction of the memory array which can be achieved [18]. With no contacts between each transistor drain and bit lines the resulting memory array can be more compact. The downside is that serially-connected transistors cause longer access times to memory cells in comparison to parallel-connected transistors.

Another way of realizing a ROM in an integrated circuit is by utilizing absence or presence of the contact between the metal layer and the gate of each transistor [18]. In this way, the transistor is present at word line, bit line cross-section but it is a metal layer, transistor gate connection which defines logic '0' or '1'. By doing this, in case of changing content of the ROM, only the contact layer has to be updated without influencing other layers. This creates an uniform array which may be (but does not have to) larger than ROM realization through the presence/absence of a transistor.

ROMs are mostly used for storing look-up tables, calibration codes, boot-up codes, and program codes for e.g. game and vending machines. The data in ROM is predefined at design stage and cannot be changed after fabrication. As a result, all design errors should be covered before the final tape-out. Generation of new masks for fabrication can be very costly.

Because ROM's storage element is based on a single transistor, scaling related relia-

bility concerns presented in *Static RAM* subsection also apply to the ROM.

EPROM

Erasable, programmable read-only memory (EPROM) is a non-volatile memory whose data can be reprogrammed. The storage element is realized through a special, modified transistor called *floating-gate transistor* [12], [56]. The modification lies in addition of another gate to the transistor - floating gate. The structure of a transistor is depicted in Figure 1.7. The floating-gate transistor consists of a control gate located at the top of the transistor. Below the control gate lies an isolated floating gate. A data in the storage element depends on the threshold voltage of the floating-gate transistor which is determined by presence or absence of a charge on the floating gate.

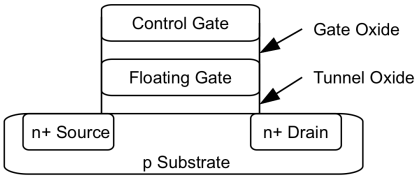


Figure 1.7: Floating-gate transistor structure.

In order to program an EPROM cell, the control gate and drain have to be biased at high voltages while the source of a floating-gate transistor has to be grounded (Fig. 1.8) [12]. A high source to drain current is run through the channel. Because of the high current, some electrons gain enough energy to escape from the channel. A positive bias on the control gate attracts electrons from the channel into the floating gate, where they become trapped. The high voltage causes the electrons in the substrate to jump into the floating gate because of a channel hot-electron injection (CHEI) process. When enough charge is stored in the floating gate, the transistor switches to the "off" state. Because of that, the programmed cell usually represents logic '0' and erased cell logic '1'. Once the charge is trapped in the floating gate, it can typically stay there for more than 10 years (data retention time).

In order to erase an EPROM cell, the EPROM has to be exposed to a strong ultraviolet (UV) light for approximately 20-30 minutes until electrons in the floating gate gain enough energy to escape from it [56]. Typically, EPROM chip have small quartz window in the package to admit UV light for erasure.

Often, it is only needed to program EPROM only once. For that purpose and to decrease production costs, chip package is devoid of the quartz window. Resulting EPROM is then called *one-time programmable ROM* (OTP ROM).

The EPROM has a limited number of times it can be updated (memory endurance). The limitation is caused by writing and erasing procedures which have negative impact on the reliability of a memory cell. More precisely, the more times electrons migrate through the tunnel oxide the bigger is the probability that some of electrons will get

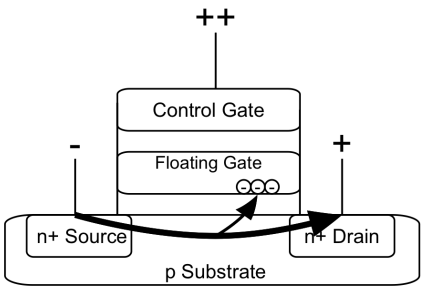


Figure 1.8: Charge injection into floating gate using channel hot-electron injection process.

trapped in it. Once an electron is trapped in the tunnel oxide it is very hard to be removed [68]. The more electrons are trapped, the harder is to maintain electrons in the floating gate. As a result, electrons stored in the floating-gate leak away decreasing data retention time and thus reliability of the whole memory.

The number of program/erase operations which a memory cell can withstand depends mostly on the way the chip is made. As a consequence, the memory endurance of EPROM can differ from manufacturer to manufacturer and is often reported in the memory's specification. Typically EPROM can be erased/programmed more than 1000 times.

Obviously, EPROM offers more flexibility than ROM when it comes to data update. Moreover, EPROM chips are mass produced and commercial off-the-shelf (COTS) memories can be used in different kinds of digital systems. Their major drawback is that in order to update data stored in the EPROM, the EPROM chip has to be taken out of the system, exposed to UV light, programmed and put back in the system.

Similarly like in DRAMs, the storage mechanism in EPROMs is based on a charge storage. Because of that, there is a possibility to control the amount of charge placed in the floating-gate transistor and store more than one bit in a single cell. While the implementation of MLC EPROM is possible, for practical reasons it is not pursued. Because of rather complex reprogrammability, EPROM technology is approaching the end of its lifetime and is being replaced by its successors: EEPROM and flash memory.

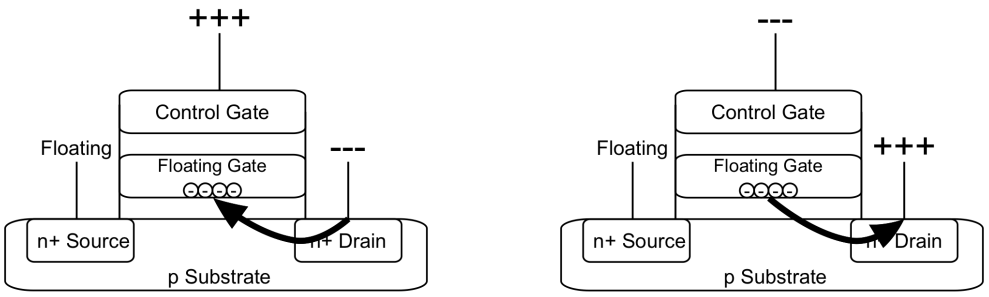
Reliability concerns related to scalability and radiation susceptibility of floating-gate transistors will be further presented in *Flash memory* subsection.

EEPROM

Due to complex re-programmability of EPROM there was a need for non-volatile memories whose data could be updated in-situ, without removing memory device from the system. This need was satisfied with the development of electrically erasable, programmable ROM (EEPROM). The working principle of EEPROM is similar to EPROM with few differences. While the storage element also consists of a floating-gate transistor, the tunnel oxide between transistor channel and floating-gate is thinner in EEPROMs [56]. Because of that, electron transportation is easier. Moreover, programming and erasing

operations are performed with an electrical approach without the need for UV light as in EPROM.

In order to program an EEPROM cell a high voltage has to be applied to the control gate and the transistor drain has to be connected to the ground (Fig. 1.9a) [56]. Due to large potential difference, electrons flow from the drain to the floating gate and become trapped. Erasing of an EEPROM cell requires opposite polarization. That is, control gate has to be grounded while transistor drain has to be connected to high voltage (Fig. 1.9b). As a consequence, electrons flow from the floating-gate to the drain and the cell becomes erased. The program/erase procedure is accomplished based on the process called *Fowler-Nordheim Tunneling*. High voltages required for programming/erasing procedures are usually generated inside the chip using charge pump circuits. Finally, the program/erase procedure is performed on a byte basis, i.e. each byte inside EEPROM can be freely modified.



(a) (b)

Figure 1.9: Charge injection (a) and removal (b) from the floating gate using Fowler-Nordheim Tunneling.

Typically, the endurance of EEPROMs is the largest from all floating-transistor-based memory devices. The number of program/erase operations typically exceeds 10^6. The downside is that the programming takes usually longer than in EPROM or flash memories. As a result, EEPROM is mainly used for reading stored data. Moreover, in order to avoid unnecessary disturbance among neighboring cells during program/erase procedures, a selecting transistor is often added to each EEPROM cell. As a result, EEPROMs are usually larger than their EPROM and flash memory counterparts.

Flash memory

A flash memory is another form of non-volatile memory based on floating-gate transistors. It is a mixture of EPROM and EEPROM technologies. The storage element consist of a single floating-gate transistor as in EPROM, but has thinner tunnel oxide just like in EEPROM [56]. Thinner tunnel oxide is required to electrically erase the content of memory cells. In contrast to EEPROMs, the erase procedure performed in flash memories

concerns memory sectors and not single bytes. As a consequence, in order to update a word in the flash memory, typically a whole sector has to be first erased. By grouping memory cells into sectors which can be erased at once there is no need for a select transistor for each memory cell as in EEPROM. Because of that, and due to thinner tunnel oxide in floating-gate transistors, the programming operation is usually shorter in comparison to EEPROMs. The downside is that erase-before-write characteristic complicates peripheral circuitry and flash memory management [32].

In flash memories, terms programmed and erased refer to injection of a charge (logic '0') into and its removal (logic '1') from the memory cell [32]. Both, injection and removal of charge requires high voltage to perform [43]. Flash memory cells can be programmed using channel hot-electron injection as in EPROM, or Fowler-Nordheim tunneling as in EEPROM. The erase procedure is performed using Fowler-Nordheim tunneling.

A typical flash memory consists of memory array, row and column decoders, sense amplifiers, high-voltage control circuits for program/erase operations and data latches and buffers [43], [32]. Based on the organization of the memory array different flash memory architectures can be distinguished: NOR, NAND, DINOR and AND. Although they are based on the same concept, they have completely different features and usage patterns [43], [32]. The most common flash memory architectures - NOR and NAND, will be further described.

The NOR architecture offers access to individual memory cells through the parallel organization of the memory array. With this structure, fast, random access can be obtained. The write operation is carried out at the word level and is much slower than the read operation. The erase operation is performed at the sector level and is the slowest operation. NOR flash memories have typically a parallel interface with separate address and data buses [47], [32]. Because of the NOR memory array organization an execute-in-place (XIP) method can be provided. Therefore, NOR flash memories are mostly used for storing the boot or the operational code and configuration files in applications where fast, random access time is required [64], [47], [66].

Due to absence of the drain contact for each cell, the NAND architecture provides smaller bit size in comparison to NOR. Because cells are arranged in series, write and read operations usually concern a memory page. Erase operation is performed on a whole memory block. The NAND organization provides faster program times and requires lower currents than NOR. The main drawback of this structure is very poor random read access which prevents from XIP ability. Usually, NAND memories contain additionally data latches and buffers for writing and reading the whole page in parallel. In NAND flash memories data and address buses are typically multiplexed on the same pins [66], [32]. Since access to the NAND flash memory is similar like in hard disks, NAND flash memories are mostly used in USB drives, memory cards, disk caches, and solid-state drives [38], [47], [66], [67].

Much research has been conducted concerning reliability of the flash technology. It is known that aggressive scaling reduces a number of electrons stored in a floating gate which causes data retention problems [32]. Furthermore, scaling the flash technology below 20 nm is projected to be very difficult [46]. In addition, decreasing cell size causes coupling

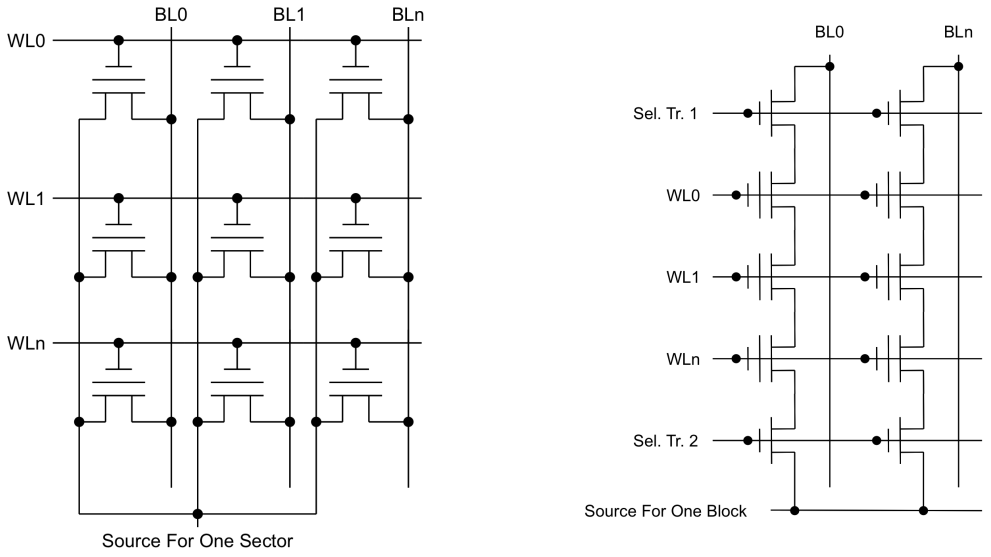


Figure 1.10: Different configurations of flash memory array: NOR configuration (a), NAND configuration (b).

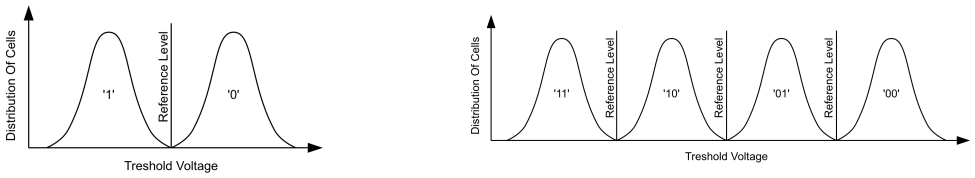
between adjacent cells and quantum effects which can change the write characteristics and influence data integrity [66].

Because a NOR cell is larger than a NAND cell, reliability of the NOR flash memory is better. NOR flash manufacturers typically guarantee that every memory cell meets retention and endurance specifications [32]. On the contrary to NOR, NAND flash manufacturers to improve yield and reduce costs can ship NAND devices with some memory blocks marked as bad. The first physical block of the NAND flash memory is assured error free and is commonly used for storing bad block tables and other information required by fault-tolerant techniques [4], [66].

Reliability of floating gate devices is a very complex problem. Since times when first commercial flash memories were available, much attention was paid to discover and model flash memory faults. As a result, mechanisms such as program and read disturbs, quantum-level noise effects, erratic tunneling, stress-induced leakage current (SILC) and detrapping-induced retention problems were discovered [67]. To simplify modeling of the flash memory, only disturbance errors like program and read disturbs are taken into consideration since they are the most frequent errors in flash memories [38]. Disturbance failures, which are not destructive, affect neighbor pages or data in a page where memory operations are performed. To mitigate the program disturb, flash manufacturers strongly suggest that pages within a block should be programmed in order [38].

Similarly to EPROM and EEPROM, the MLC concept can be implemented in flash technology. In MLC flash memories, each cell stores specified amount of a charge which can be sensed by many reference levels (Fig. 1.11). MLC flash memory has lower relia-

bility than SLC [38]. Typical endurance of the MLC flash memory is 10^4 program/erase operations in comparison to 10^5 for the SLC. In addition, MLC devices have on average longer and enormously variable program latencies which depend on data values to be stored [38]. They consume more energy for read and program operations and also more idle power than SLC memories.



(a) (b)

Figure 1.11: Floating-gate transistor treshold voltage distribution in SLC (a) and MLC (b).

Besides that flash memories are susceptible to complex defects and process variations, they are also prone to radiation induced failures – single event effects (SEEs). Although they are not as vulnerable to failures induced by SEEs as SRAMs and DRAMs, high energy particles can influence their content because of the charge-based storage elements [6]. Despite soft errors observed in a memory array, the most exposed parts of the flash memory are charge pumps and internal circuits [47]. After specified dose of radiation, a memory loses ability to program or becomes damaged.

In general, the NOR memory seems to be less susceptible to single-event functional interrupt (SEFI) errors than the NAND device due to simpler internal circuits. It is also worth to notice that cell's radiation sensitivity depends on its threshold voltage. A memory cell that has been programmed with fewer electrons is more prone to soft errors [6].

1.1.2 Emerging non-volatile memories

Phase-change RAM

History of memories based on phase-change materials started in the sixties. At the beginning they did not gain much attention because of the poor performances and capabilities of first devices. After discovery of new materials, especially combination of Germanium, Stibium, and Tellurium (GST), phase-change memory started to gain new interest [32].

Phase-change random access memories (PCRAMs) are based on the unique properties of chalcogenides, materials which can exist either in an amorphous or a crystalline form at room temperature. The amorphous phase exhibits high resistivity in contrast to the crystalline phase [32], [20]. Storage element consists of a phase-change material and electrodes with a resistive element acting as a Joule heater (Fig. 1.12). The Joule heater is used to change the phase of the chalcogenide material. Reading is performed by running a low current through the memory cell and measuring its resistivity. Programming is achieved by running a high current through a cell, heating it up, and then cooling it

quickly to leave it in the amorphous state called RESET (logic '0') or slowly to allow the crystals to grow and place the cell in the SET position (logic '1') [102], [99], [46], [20]. The program operation requires higher than the nominal supply voltage and draws significantly more current and power than the read operation [102], [20]. Therefore, similarly like in flash memories, charge pumps are needed. To limit instantaneous current level, some PCRAM prototypes support iterative writing of smaller data units than a memory word [20].

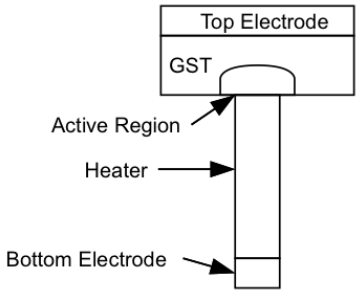


Figure 1.12: Structure of the PCRAM cell.

At the array level, PCRAM storage elements cannot be directly connected to word and bit lines. The reason is the leakage path which can be formed through conductive storage elements. As a result a typical PCRAM cell consists of a storage element accompanied by a select device.

PCRAMs are byte addressable and in comparison to flash memories they do not require a block erase before the write operation [27], [32]. The phase-change technology offers excellent scalability with current CMOS process fabrication methodologies [104], [99]. It is projected that the PCRAM scaling mechanism will be more robust than that of the DRAM beyond 40 nm [82]. Moreover, program latency and power shrinks as memory cells scale down [55], [81]. The critical concern connected with PCRAMs scalability is related to the size of the select transistor. Because the reset current is substantially high, the minimum width of the transistor able to drive required amount of current is limited [80].

Since rebirth of the phase-change memory, its reliability was heavily studied. It was shown that program and read disturbs are not a concern at today's technology nodes [79], [81], [35], [46]. One possible problem could be related to data retention. The amorphous phase of the GST is a meta-stable phase in comparison to a stable, crystalline stack [46]. While spontaneous crystallization at room temperature could take hundreds or even thousands years to occur, at elevated temperatures, crystallization rate increases exponentially [79]. To mitigate this problem a refresh operation performed every few days, months or even years, depending on the environmental factors, could be sufficient.

A PCRAM cell is expected to sustain 10^8 program operations at the 65 nm node before it breaks and manifests as a stuck-at fault [35], [46]. This number strongly depends on the manufacturing capabilities and process variations. In case of 4MB C-RAM which is a fabricated, radiation hardened phase-change memory, ensured endurance is only 10^5 [79].

This big discrepancy is thought to be caused by immature fabrication processes where preparation of high quality GST and heating elements is not yet reached. Although the number of program operations is projected to be greater than in case of the flash memory, it has to be further increased to allow the PCRAM to be usable as a main memory. Moreover, as it was shown in [79] temperature and data pattern have significant impact on memory endurance. Fortunately, program operations do not influence retention time like in case of the flash memory [35].

Because there is a large resistance difference between SET and RESET states in the GST, there is a possibility to implement the MLC concept in a PCRAM cell. With advanced programming operation it could be possible to change a phase of the chalcogenide material to be partially amorphous and partially crystalline [55]. The main obstacle preventing from development of reliable MLC PCRAMs is a time dependent resistance drift in amorphous chalcogenide materials [57]. That is, different resistance levels stored in the cell drift up towards high resistance values over time. As a consequence, the read value of a PCRAM cell can change over time.

Unlike charge-based memories like the DRAM or the flash, PCRAM cells are not prone to radiation induced errors at current feature size [79], [81]. Although, it is important to remember that even though the memory array is robust and is not susceptible to radiation, the peripheral circuit may be not [32], [46].

With non-volatility, considerably low power consumption, high endurance and good scalability, the PCRAM is expected to replace the flash memory or even the DRAM in the near future [102]. Moreover, with different memory array organizations, PCRAM is foreseen as NOR flash memory replacement in embedded applications and as NAND flash replacement in storage class memories [28], [26].

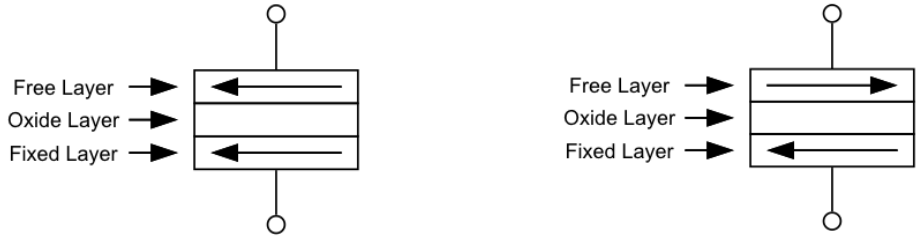
While PCRAM product demonstrators were already presented (e.g. 128 Mb SLC PCRAM [72] and 256 Mb MLC PCRAM [7]), still a lot of research and development is needed to reach fully mature fabrication level.

Magnetoresistive RAM

First magnetic memories were introduced in the forties. In the late eighties Giant Magnetoresistance (GMR) was discovered leading to increased interest in using it to build a semiconductor non-volatile memory. Many years of research resulted in a 128 Kbit magnetoresistive random access memory (MRAM) test chip in 2003 [9]. Since then, a lot of research was conducted in order to characterize magnetoresistive memories and improve their fabrication.

In MRAM, a storage element called *magnetic tunnel junction* (MTJ) consists of a thin oxide layer sandwiched between two layers of ferromagnetic materials (Fig. 1.13) [56]. The top ferromagnetic layer (Free Layer in Fig. 1.13) represents a soft magnet and is used to store information. The bottom ferromagnetic layer (Fixed Layer in Fig. 1.13) has a fixed magnetization and is used as a reference magnet. Based on the magnetization of ferromagnetic materials which can be in parallel (Fig. 1.13a) or anti-parallel (Fig. 1.13b) configuration, the resistance of the MTJ is low or high, respectively [32]. Writing information to MRAM cell requires running an electrical current in electrodes in orthogonal

directions. A magnetic field produced by running current changes a magnetization of the soft magnet. In order to reliably store data in the MRAM and reduce cross-talk between MTJ's each storage element also requires a select transistor.



(a) (b)

Figure 1.13: MTJ in parallel (a) and anti-parallel (b) configurations.

The main benefits of MRAM are non-volatility, fast write and read access times (comparable to that of the SRAM) and practically unlimited endurance. Moreover, similar like in the PCRAM, it offers byte addressability and does not require a block erase before the write operation as in flash memories.

Although MRAMs offer great characteristics, the major challenges preventing them from wider adoption are related to their power-consumption and scalability. That is, high currents required for generating magnetic fields for writing are non-scalable [10]. In addition, as the MTJ dimensions shrink, MRAM cell thermal stability is reduced resulting in data retention problems and bit errors.

Solution for this problem can be an MRAM architecture where program operation can be realized through spin-torque transfer (STT) technique. The main idea is based on forcing a spin-polarized current through the MTJ instead of using a magnetic field to perform the program operation [5], [74]. This approach lowers the amount of current needed for programming and facilitates scalability of the MRAM [32]. MRAMs which are programmed with spin-torque transfer technique are referred as STT-MRAMs.

As the STT-MRAMs are gaining more interest and research focus, several challenges have to be addressed concerning their scalability. Even though STT-MRAMs scale better than MRAMs, at smaller feature sizes process variations coming from magnetic devices and CMOS technology may have a negative impact on the reliability. For example, doping and geometry variations of select transistor can have impact on the switching current running through the MTJ. Moreover, variations concerning the oxide layer and cross-section area of ferromagnetic layers can severely impact characteristics of the MTJ. As a result, cell failures can occur where e.g. read operations can influence data stored in the STT-MRAM cell or the current running through the MTJ can be insufficient to store data during program operation.

Through modification of the MTJ or adding a second serially connected MTJ to a memory cell, an MLC STT-MRAM can be realized [56]. In [60] authors demonstrated MLC MTJ device able to store two-digit information ("00", "01", "10", and "11") in

four resistance levels.

There are no known radiation effects for ferromagnetic materials. Combining this characteristic together with unlimited endurance [23] makes MRAMs and STT-MRAMs attractive storage devices for aerospace applications. Moreover, with non-volatility and access times comparable to that of the SRAM, magnetoresistive RAMs are foreseen as SRAM replacement in field-programmable gate arrays (FPGAs) [14], [108] and cache memories [49], [22], [70].

Ferroelectric RAM

Ferroelectric random access memories (FeRAMs) are non-volatile memories which combine low-power consumption, fast read and write times (one order of magnitude faster than flash), and significant endurance (six orders greater than flash) [29].

The FeRAM architecture is similar to DRAM, i.e. the memory cell consists of a capacitor and a select transistor. The difference is that while DRAM capacitor is based on linear dielectric, FeRAM capacitor includes ferroelectric material in the dielectric structure [86]. FeRAM cell can be configured in two polarizations: parallel and anti-parallel. Writing is performed by using a direction of an electric field applied to the storage element in order to change the polarization. Reading is performed by applying an electrical field with predefined direction (e.g. direction which forces memory cell to '0' state) to the storage element. When predefined direction (e.g. used to write '0' to the memory cell) of an electrical field matches direction of storage element's polarity (parallel) then no pulse is generated on the bit line, and cell's state is assumed to be '0'. When an electrical field is in different direction than the storage element then a pulse is generated on the bit line and cell's state is assumed to be '1'.

Typical endurance of FeRAMs surpass the endurance of other non-volatile semiconductor memories (except MRAMs). But similarly like in DRAMs, read-outs are destructive, i.e. once the memory cell is read, the cell's original value has to be restored. As a result FRAM's endurance is approximately 10^{12} - 10^{15} read/write operations [107], [45], [25]. In contrast to DRAMs, FeRAMs do not require refresh operation.

While densities compared to DRAM should be possible to achieve with FeRAMs, further material engineering is required in order to reach ultra-high densities. Even though commercially available FeRAMs reach only up-to 4Mb capacity they have found applications in smart cards [86], aerospace [62], and automotive markets. Moreover, they have also been implemented in microcontrollers where they replace both SRAM and flash memories [45], [30].

1.1.3 Conclusion

To fulfill growing requirements of today's applications, chips are moving from logic-dominant to memory-dominant ones. As a result, the SoC yield and reliability relies heavily on the yield and reliability of memory cores embedded into a system [36], [64].

Scalability related problems of conventional semiconductor memories are forcing researchers and system designers to seek for new memory technologies. The final goal is to

find a memory which would fulfill all assumptions of the universal memory concept. Although there are well established and mature non-volatile technologies like the EEPROM or flash, it is unclear if circuit and technology improvements will enable future viability of these technologies. The main concern is the charge-based storage mechanism. As feature size decreases the precise placement and control of a reduced amount of charge is expected to be severely difficult [55], [81]. Fortunately, new memory technologies are being developed. Among them, PCRAMs, MRAMs/STT-MRAMs, FeRAMs, and RRAMs offer the most promising characteristics [82]. One of them is that they are not established on charge storage. Instead, they are based on changing arrangement of atoms which offer greater potential in scalability. Further, they are byte addressable, offer XIP, and show greater performance and endurance than the flash [46].

However, we cannot forget about their disadvantages. Despite intrinsic problems related to the technology or process variations, emerging non-volatile memories exhibit asymmetric behavior where program operations are usually longer and consume more time and energy than reads [104], [46]. Necessity for the high current can also lead to such problems like e.g. electromigration in metal lines which can limit their scalability [32]. In addition, due to unique storage mechanisms, they require specialized circuits for performing complex program operations [104]. Errors in the peripheral area can mask real, unique types of failures, making memory usage and testing more difficult. Furthermore, complex peripheral circuitry can be more vulnerable to radiation than the memory array [47], [6].

What is also worth to notice is that manufacturers delivering product specification may not provide all the required information. In fact, real performance of the device can be worse and highly variable [38].

Finally, while each of market-available PCRAM, MRAM, and FeRAM poses its own peculiar features, none of them fits completely into the universal memory concept [12]. For example, FeRAM has the lowest power consumption but is difficult to scale. MRAM has the fastest access times but it requires most current for programming. PCRAM offers best scalability but has the lowest endurance. As a result, it may turn out that none of emerging memories will become the universal one and other technologies have to be developed. On the other hand, it may be that system-level solutions and/or advanced management mechanisms will expand the capabilities of emerging memories to the point they will become universal.

1.2 Non-volatile memory management

As outlined in the previous chapter, storage mechanisms in non-volatile memories are usually more complex than those found in, e.g., Mask ROMs, SRAMs, or even in DRAMs. As a consequence, NVMs often require special management which properly handles their peculiarities such as:

- limited endurance,
- asymmetric access,

- technology-, endurance-, and retention-related faults,
- vulnerability to environmental factors (e.g. elevated temperatures, magnetic field), and/or
- read and write disturbs.

In addition, another aspects which may require special attention are problems related to decreased feature size (e.g. like in flash memories) or immature technology (e.g. like in emerging NVMs). Therefore, NVM management techniques are focused mostly on reliability concerns.

Many NVM management mechanisms are based on already known techniques which were developed for SRAMs or DRAMs (e.g., coding techniques, redundancy repair[7]). Some of them were created specifically for NVMs (e.g., wear-leveling mechanisms). Further in this chapter, the most common and interesting NVM management solutions will be described.

1.2.1 Endurance improving techniques

Most of the existing and emerging NVMs suffer from limited endurance. Depending on the memory technology, array organization, quality of fabrication process, and/or the feature size, the number of destructive[8] memory operations which can be reliably performed on the memory varies significantly. Furthermore, the endurance of non-volatile memory may be influenced by external factors such as supply voltage or ambient temperature. On top of that is the memory usage pattern which has one of the biggest impacts on the memory lifetime. For example, few program operations performed once a day will result in different memory lifetime span than rewriting whole memory every few minutes. Since the lifetime of a single word, row, or memory block depends on the reliability of a single cell, it may be crucial to implement solutions which will reduce or evenly distribute the wear of memory cells [55].

One of these solutions, called *wear-leveling*, is based on managing even distribution of program operations across the memory. Originally, wear-leveling techniques were introduced for NAND flash memories. Due to the block erase requirement and the block-based access of NAND flash memories, most of developed wear-leveling techniques are based on logical-to-physical block mapping [76].

There are two types of wear-leveling techniques called *dynamic* and *static* [66]. In the dynamic wear-leveling if a logical block has to be rewritten, its new data is stored in a different physical block. A block with original data is marked as invalid and the mapping table is updated [31]. In this approach static data which is not frequently updated is not

[7] Although many types of redundancy exist (e.g., time, information, hardware), the term *redundancy repair* used in the thesis refers to repair mechanisms based on exchanging corrupted memory array areas with spare ones.

[8] Memory operations which decrease the endurance of the memory. In PCRAMs, RRAMs, and flash memory, the memory write/erase operations are destructive. In FeRAMs, both, memory write and read operations are destructive.

moved and the wear mechanism affects only dynamically updated parts of the memory. The static wear-leveling is similar to the dynamic with one exception: in this approach static data is periodically moved to different locations in order to evenly wear out the whole memory area. After some time, the number of unused blocks can be lower than a certain threshold. In that case blocks marked as invalid have to be erased in order to reclaim free space. This operation, called *garbage collection*, can be executed during memory idle time or on demand [31], [66].

Mapping techniques based on writing modified blocks to variable positions distribute wear-out of memory cells evenly. Moreover, writing to another part of the memory releases the user from immediate block erase, postponing it for later. Another interesting advantage is that if during the write operation the power is lost, the previous state of the block can be recovered [31]. The disadvantage is that managing mapping tables and counters for tracking the number of destructive operations requires a large amount of storage overhead which scales with the memory size. Moreover, large tables impose large look-up latency which can have a negative impact on memory performance [76].

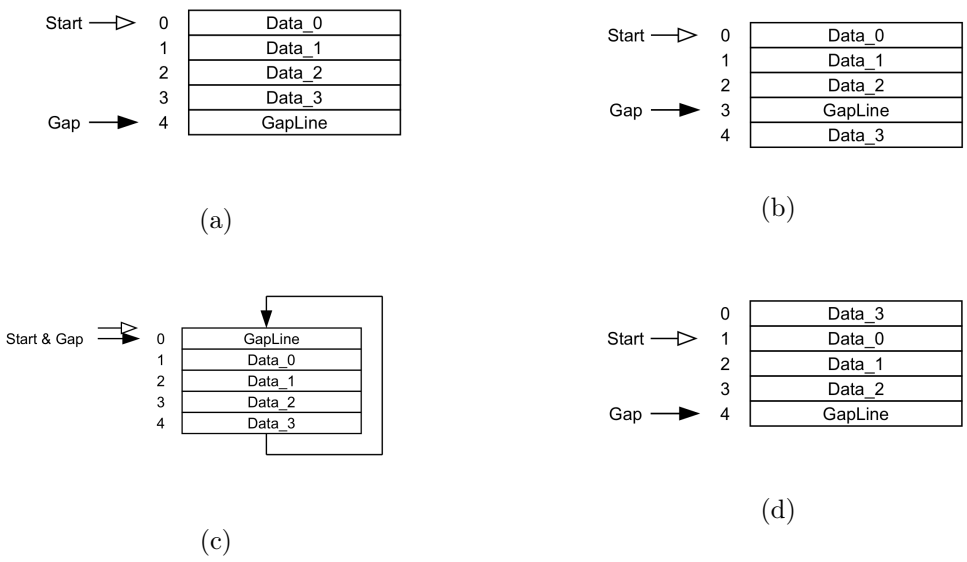


Figure 1.14: Start-Gap wear leveling implemented in the memory containing 4+1 rows. Figure adapted by author from [76].

Tables can be eliminated if an algebraic mapping instead of direct mapping is used. Based on this idea, a Start-Gap technique was introduced in [76]. To provide wear-leveling, the Start-Gap requires an additional memory row called *GapLine* and two registers called *Start* and *Gap* (Fig. 1.14a). The Start register initially points to 0^{th} memory row and the Gap register always points to the location of the GapLine. Initially, the additional memory row is assumed as GapLine. After a predefined number of writes to the memory, the content of the row addressed by the [Gap-1] is copied to the GapLine and the Gap register is decremented (Fig. 1.14b). This procedure is continued for all

memory rows. If Gap register points to 0^{th} memory row then [Gap - 1] address points to the last memory row similarly like in a circular buffer (Fig. 1.14c). Next, when all rows in the memory have been shifted (Fig. 1.14d) the Start register is incremented. Based on the Start and Gap registers, the logical-to-physical line address mapping is performed. It was shown in [76] that the basic Start-Gap algorithm can extend the memory lifetime by 53%. After introducing randomization to the algorithm, the endurance can be increased even to 97% of the ideal wear-leveling concept.

Another group of memory endurance improving solutions are techniques based on reducing the write traffic to the memory. They can be divided into two categories. The first category is exploiting the external fast memory with unlimited endurance (e.g. SRAM, DRAM) as a cache memory. The second category of solutions, developed for byte addressable memories which does not require erase-before-write (e.g., emerging NVMs), is based on avoiding unnecessary write operations. Since the mechanisms using cache memories are already well known and heavily used, only the latter one will be further described.

As mentioned, write operations in NVMs usually have negative impact on memory's reliability and are expensive in terms of power consumption. If the number of memory write operations could be reduced, then the lifetime of the memory could be extended and average power consumption decreased. The simplest approach to achieve this is to read the memory content before the memory write operation, compare it with the input data, and update only information which differs. This concept reduces power consumption, improves the bandwidth, and extends memory's lifetime [102]. The main drawback is a need for the read operation before the block can be updated. But since write operations are more time consuming than read operations, the additional latency is comparatively small.

Another similar technique called Flip-N-Write was presented in [20]. The main idea of this concept is based on the read-modify-write procedure with additional flipping of entire word to reduce the maximum number of bits to be programmed. That is, before an input word is stored, first, the location where the word should be stored is read. Next, the output word is compared with the input word and its inverted version. The version of the input word (original or inverted) which requires less bit changes in the stored data is written to the memory. The drawback of the Flip-N-Write is that it requires an extra bit per word for indicating whether the stored word was flipped or not.

In [102] a different idea was presented. To reduce the number of unnecessary bit writes, memory blocks which are no longer used are indexed using a content-based signature. Next, when new data is about to be written, its signature is compared with the signatures of unused memory blocks. An unused memory block whose content is the most similar to the content of the input data is selected for storage.

Apart from technology limitations, it is the memory's usage pattern which has significant impact on memory's endurance. Therefore, for each specific application, worst-case scenarios or even malicious security attacks should be taken into consideration while designing a digital system which employs NVMs. Especially the possibility of malicious security attacks should be verified in advance. If the attacker knows the wear-leveling

mechanism and a way in which it can be exploited then the accelerated system damage can be caused. This problem was addressed in [85], [76], [54], [83].

1.2.2 Redundancy repair

The first semiconductor memories were made testable and not repairable. But since they started to dominate SoCs area it was crucial to provide some kind of repair mechanism in order to improve system's yield [64], [51]. The most utilized technique used for memory yield improvement is a redundancy repair. Standard memory repair using redundancy consists of testing the memory with an automatic test equipment (ATE) or by a built-in self test (BIST). Next, based on the error distribution, the host computer of the ATE performs redundancy analysis (RA) to efficiently allocate spare elements. Further, appropriate electrical fuses are blown or a fuse box consisting of non-volatile elements is programmed to deactivate defective memory areas and to activate spare elements [71], [95], [36], [43], [37], [34], [51]. This approach is becoming too time consuming for today's large embedded memories and turns into a complex problem due to limited ATE I/O bandwidth. Additionally, since this method is based on heavy usage of the test equipment it can take up to 40% of the manufacturing costs [71], [64].

To address these issues a built-in self repair (BISR) is considered as the most cost effective solution [42], [43]. The typical BISR design consists of three main blocks, the BIST, a built-in redundancy analysis (BIRA) and an address reconfiguration (AR) module (Fig. 1.15) [43], [44]. The BISR design depends mostly on the BIRA module which tries to effectively allocate spare elements. Both, the hardware complexity and the time penalty are affected by the number and types of spare elements. Many RA algorithms have been developed for RAMs, but new NVMs require special architectures which are more complex [42], [43]. There is no universal and cost-effective RA algorithm which could be used for different types of memories and different types of spare elements [44].

There are several possible types of redundant elements in the memory array like bits, words, blocks, rows, and columns which can be used for repair [71], [42], [95], [36]. Small block/bit based allocation can be more effective for replacing defective memory areas than techniques based on row/column replacement. But the drawback of this solution is increased redundant area and more complex AR circuitry due to bigger remapping tables [71], [43], [27]. Using only row or column redundancies 100% of repair rate can be hardly realized. With 2-D (both, row and column) redundancy 100% can be achieved but the main drawback is that the 2-D redundancy analysis problem is NP-complete, resulting in complex RA algorithms [42], [43].

After the RA procedure, the information about exchanged elements can be stored in a content addressable memory, the NVM or the SRAM [43], [44]. This information has to be read on every access to the memory to remap resources. Therefore, the access time to this information and the time penalty of the AR circuit are crucial for memory performance [95], [44].

There are many architectures and variations of BISR designs which offer different repair mechanisms. Most of them are off-line redundancy repair techniques. That is, the repair

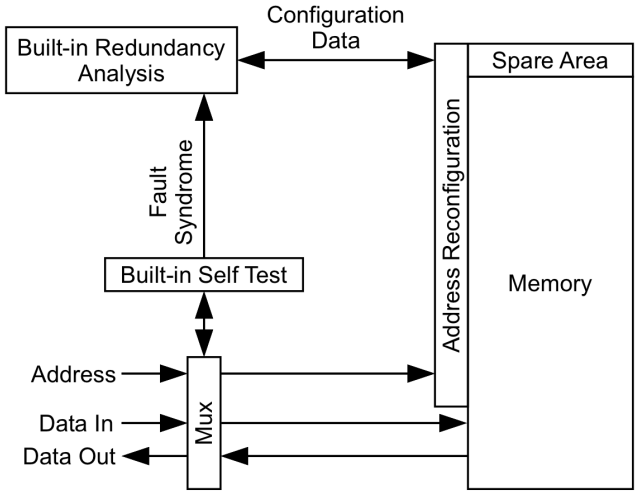


Figure 1.15: Block diagram of a typical built-in self-repair design.

procedure is usually performed once, right after production in order to improve the yield of the memory. But with the introduction of NVMs and endurance-related faults which can occur during memory lifetime, an on-line redundancy repair mechanisms are gaining interest. Further, only selected on-line redundancy repair techniques are described.

In [81] a technique using bit-based redundancy repair is proposed. The main principle lies in exchanging corrupted memory cells in each row with spare cells. This is done by adding an error-correcting pointers (ECPs) to every memory row. Each ECP is able to store address of a single corrupted cell within the memory row and provides a single replacement cell [81]. In order to obtain the positions of corrupted bits, the repair mechanism based on ECPs uses a write-verify scheme. That is, after each write operation to the memory, the memory read operation is performed. Next, the input and output data are compared. Any differences between the input and output data are considered to be caused by wear-out memory cells. An example concerning ECP technique able to replace 5 corrupted cells in 512-bit memory row is depicted in Figure 1.16 and 1.17. Figure 1.16 shows the required layout of the memory row where in addition to the 512-bit user area, 5, 10-bit ECPs are needed. Besides ECPs, another bit is required (*Full* bit) to mark the situation where all ECPs are used [81].

Figure 1.17a shows the memory row after write operation where the verify procedure reported a single corrupted cell at 1^{st} position. After detecting a fault, the first ECP (ECP0) is filled with the position of the corrupted bit and the replacement bit (marked with 'R' in Fig. 1.17a) is filled with the value of the 1^{st} input bit. The last ECP (ECP4) contain a unary-encoded counter denoting how many of the other ECPs (ECP3 to ECP0) are active.

In case when fault occurs in the used ECP, an unused ECP with higher index is assigned for the correction of the fault corrected by the corrupted ECP [81]. For example, in Figure

1.17b the replacement cell in the ECP0 has failed. To compensate for this, ECP1 is activated and filled with the position stored in the ECP0. The replacement cell in ECP1 supplants now both, the original failed cell and the failed replacement cell in ECP0. In the ECP-based repair, ECPs with higher indexes have precedence over ECPs with lower indexes [81].

The disadvantage of the ECP-based technique is that it requires a special design of the memory array. That is, each memory row has to be extended in order to accommodate area required for repair. Moreover, as pointed in [81] an additional circuitry is required inside the memory chip. As a result, interference with memory design process may limit the applicability of the approach and impact memory's performance and testability.

Figure 1.16: Layout of a memory row where ECP5 is implemented. Figure adapted by author from [81].

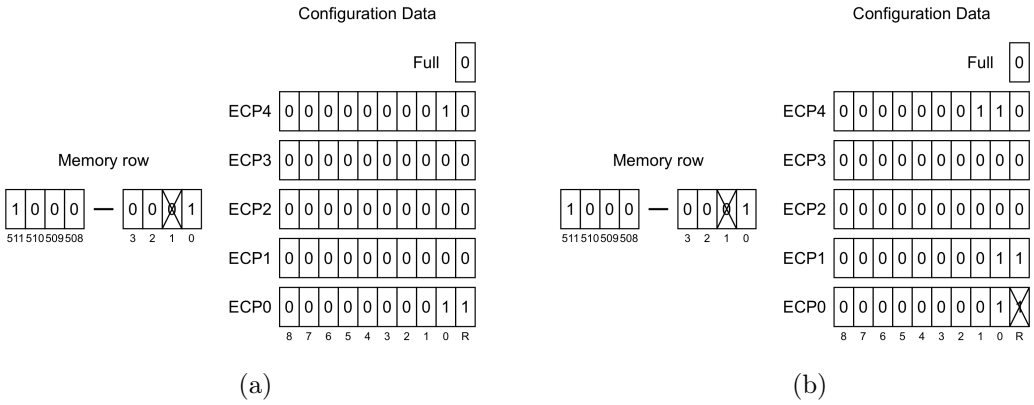


Figure 1.17: Example of the ECP-based repair mechanism correcting a single-bit fault in the memory row (a) and second-bit fault in the used ECP (b). Figure adapted by author from [81].

In [84] a technique offering an alternative approach for recovering from stuck-at faults is presented. A mechanism called *SAFER* exploits the fact that a failed cell with a stuck-at value is still readable and can be used to store data. SAFER provides a dynamic partitioning of data blocks ensuring that each partition has at most one failed bit. Further, data in the partition is stored in the original or inverted form[9] in order to match the stuck-at value of a failed cell.

The idea of partitioning is further described using an example similar to the example provided in [84]. Figure 1.18 shows an example data block consisting of 16 bits which can be partitioned into four groups (each group is depicted with different color). The

[9]Similarly to the Flip-N-Write technique described in Section 1.2.1.

partition mechanism assigns a group index for each bit in the data block using two bits from the bit pointer. For a data block composed of 16 bits the bit pointer is 4 bits wide. This implies that there are $\binom{4}{2} = 6$ ways to choose two bits out of 4-bit pointer [84]. Based on the fail-bit locations, one of these six possible ways is chosen to determine the four groups.

In Figure 1.18, the initial partition arbitrarily uses the second and the third LSBs. For each data bit the concatenation of these two bits in its bit pointer represents its group index. The partition fields indicate which bit positions are used for partitioning the data. In the example, the two partition fields indicate that the second and the third LSBs are used from the bit pointer. The fixed partition counter keeps track of the number of partitions that are fixed. Because in Figure 1.18 there are no faults, the value of the fixed partition counter is zero.

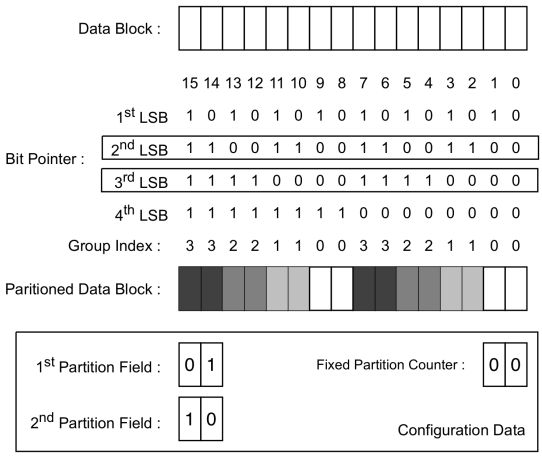


Figure 1.18: An example showing the initial state of the partitioning mechanism used in SAFER. Figure adapted by author from [84].

Figures 1.19a, 1.19b, and 1.19c show how a partition can be changed dynamically to account for new faults [84]. If the first corrupted bit occurs at bit position 9 (Fig. 1.19a), the initial partition is still valid because none of the groups has more than one fault. Now, if the second corrupted bit occurs at bit position 8 (Fig. 1.19b) then there are two faults in group 0. Thus, a new partition should be derived so that the two faults are in different groups. The difference vector of the positions of the two faults is ("1001" $\oplus$ "1000" = "0001"), which implies that the first LSB should be used for the first partition field. Thus, the first partition field is set to "00" (Fig. 1.19b) and the fixed partition counter is increased by one to account for fixing the first partition field. After this repartition, groups 0 and 1 each have one fault.

If the third corrupted bit occurs in position 2 (Fig. 1.19c), the group 0 will have two faults. Because of that, another repartition is required. The difference vector of the positions of the two faults is ("1000" $\oplus$ "0010" = "1010") which implies that the second and the fourth LSBs are candidates for the second partition field. Here in the

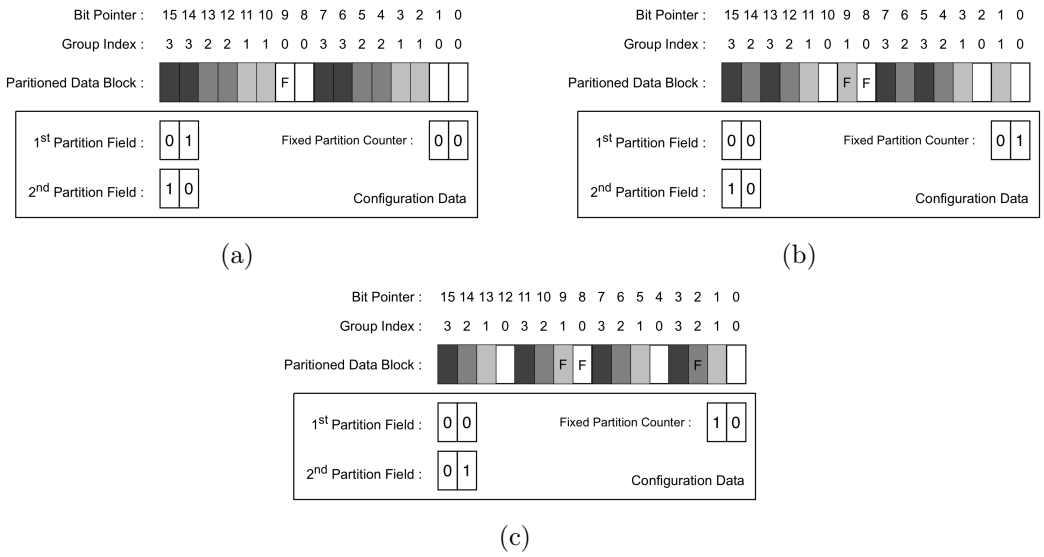


Figure 1.19: Example of the partitioning mechanism used in SAFER. Figure adapted by author from [84]

example the second LSB was used for the second partition field (Fig. 1.19c). After repartitioning, groups 0, 1, and 2 each have one fault. Furthermore, the fixed partition counter is incremented and reaches its maximum value of two, making it impossible to repartition further. Thus, in this example, data block can be recovered from a fourth fault only if the corrupted bit occurs in group 3.

In addition to partitioning, SAFER technique utilizes data block inversion for each partition to match the value of the stuck-at cell with the value of the input data. As a consequence, additionally to the fixed partition counter and partition fields, also a single bit for each partition is needed in order to mark if the input data was stored in the original or inverted form.

Similarly to ECP-based technique, SAFER mechanism also requires write-verify scheme in order to determine new faults in the data block. Moreover, a multi-bit fault repair process may consist of several write-verify and repartitioning procedures. Nevertheless, the biggest disadvantage of the approach is the complex repair mechanism which may be hard to realize in the hardware.

In [46] the idea of replicating a single physical page over two fault pages is explained. The dynamically replicated memory (DRM) method reuses memory pages that contain hard faults by dynamically forming pairs of complementary pages that act as a single page. This idea is based on the assumption that there is a very high probability that errors in different pages occur on different positions.

In the DRM each byte in the memory has its own fault-indication bit. In case when a new fault occurs in a byte, the indication bit in that byte is set. Next, the operating system marks the page as corrupted and adds the page to the list of faulty pages. Further, during page allocation performed by the operating system, a pair of compatible faulty

pages is selected. Faulty pages are compatible only when faults in one page are at different positions than faults in the other page. Next, the paired pages are treated as a single valid page. Although this solution provides dynamic repair during the lifetime of the memory, the memory storage area is significantly reduced even when pages contain only single bit failures. Moreover, the DRM requires the support from the operating system which limits the applicability of the approach.

Although the increasing number of spare elements can lead to a very high repair rate, every BISR technique has to be designed very carefully and precisely. A large number of redundancies can increase the BISR complexity and area, which can in fact decrease the overall yield and reliability of the whole system [43], [44]. Moreover, with the development of emerging NVMs an increased research focus is required in the area of on-line redundancy repair techniques. According to the ITRS, the BISR will soon become a must for yield improvement in SoC designs [46].

1.2.3 Error-correcting block codes

There are many factors which can influence consistency of data stored in the NVM. Besides errors caused by technology issues like disturbs, retention and endurance problems, there are external sources like electromagnetic contamination and particle radiation [50], [66]. Cosmic rays, solar activity or radioactive elements are not only a problem for space applications but also in aeronautics or even at the ground level [6].

Memory's radiation sensitivity depends on the technology, the feature size and the architecture. While memory cells typically take the major part of the die area, the rest of the memory chip is used for peripheral circuits [83]. Even if the memory array can be very robust against radiation, the additional logic may be not [32].

Error-correcting codes (ECCs) are very powerful for protecting memories against soft errors [33], [50], [17]. The main principle of the repair is based on generating redundant information which can be used in case of occurrence of errors to restore the original, uncorrupted data. The redundant information is created according to the data which should be protected. In a practical implementation, during memory write operation a special redundant bits (parity bits) are filled with appropriate values which are based on the input data. Further, input data together with the parity bits is stored in the memory. Next, when data is read from the memory, first, parity bits for the output data are calculated and compared with the parity bits read from the memory. In case of a mismatch, information from comparison as well as read parity bits are used to locate and fix occurred errors.

There are few classes of codes such as Hamming, Reed-Solomon (RS), and Bose-Chaudhuri-Hocquenghem (BCH) which are best suited for memory systems [50], [37]. To choose an appropriate ECC several important factors have to be taken into account. First of all, knowledge about error rates and types is required [50]. Hardware complexity strongly depends on the correction capabilities of the code. Choosing stronger error protection than needed will lead to ineffective hardware usage, while insufficient error correction level will result in decreased reliability. Information about error types can be helpful in selecting the appropriate ECC, since the optimal codes for burst errors and

random errors are different. Another important issue is data size. There is a tradeoff between parity/data ratio and hardware complexity and latency. A larger data block size provides lower relative redundancy but results in longer encoding and decoding latencies [13], [21]. The last steps in choosing the appropriate ECC are the determination of the code, its parameters, the design of encoders and decoders, and the implementation which is usually done in hardware [50], [66].

ECCs correcting a single-bit errors such as Hamming codes are used in low latency memories like SRAMs, DRAMs, or NOR flash [79], [2], [35]. Those memories require a very fast decoding and encoding procedures. Moreover, to speed up the decoding process, the parity bits are usually stored together with the data bits. Thus it is important to have a low number of bits required by the ECC [37].

For correcting multi-bit errors stronger codes such as RS or BCH are required. RS and BCH codes are frequently used in large-capacity memories (e.g. NAND flash). Although they have similar structure, RS codes are better suited for correcting burst errors while BCH codes are more efficient in random faults [21], [66], [53], [2]. In addition, because BCH codes require fewer parity bits, they are more frequently chosen for protecting SLC NAND flash memories [13], [17]. For MLC NAND flash, RS codes with similar code rate and length as BCH codes can achieve similar error-correcting performance with lower decoding complexity as presented in [15].

Multi-bit error correction is usually implemented for block-based memories where read and write units are a few hundreds of bytes in size. While encoding usually takes only few cycles of latency, decoding and correcting phases can last for a large number of clock cycles. Delay provided by the ECC is less noticeable in comparison to latencies caused by reading or writing large blocks of data [66]. Moreover, stronger error correction requires a larger number of parity bits. To keep the parity/data ratio relatively small, large blocks of data are needed.

There are several ways to speed up the decoding and encoding processes [21]. One is to increase the clock frequency for the encoder and decoder modules. The big drawback of this solution is the circuitry which is hard to realize and consumes much power. Another solution is to use multiple error-correction units. While the need for higher clock frequency is relaxed, the parity/data ratio increases due to the smaller data blocks managed by a single unit. Another solution is based on applying parallel transformations to the ECC algorithm if possible. By using parallel transformations more data bits can be processed in a single clock cycle.

In [21] BCH architectures with different characteristics in the code rate, hardware complexity, and power consumption are presented. To improve the reliability of the NAND flash memory whole-page, sector-pipelined, and serial BCH architectures are proposed. The whole-page architecture processes one page using the BCH code with the parallel algorithm transformation. The sector-pipelined architecture is also based on a parallel BCH algorithm using less complex error-correcting circuitry for managing smaller data blocks (sectors) in a pipelined way. The last architecture is based on a serial BCH algorithm employing several error correction units for smaller data blocks. It was shown in the paper that the whole-page architecture is the best in terms of the code rate.

The sector-pipelined architecture demands the least complex hardware and lowest power consumptions among all.

Another solution presented in [100] offers a BCH decoding algorithm which employs a division-free transform and combines arithmetic operations. By using these improvements the decoding latency can be significantly reduced. As shown in the paper, a 4-bit BCH decoder for 2-bit per cell MLC NOR flash memory achieves only 6.4 ns of decoding latency at 180 nm technology node.

While ECC can correct a limited number of errors, there are techniques which can improve error correction in general. In [36] a solution based on the Hamming code and a threshold voltage analysis is presented. In situations where a 2-bit error occurs in a word, two additional read operations with different voltage levels are performed. By reading the same word with different voltage levels, strong and weak bits can be analyzed. Based on this information weak bits in a word can be corrected reducing the number of errors to the number manageable by the ECC. This technique is applicable only in situations where at least one of bit errors is caused by a weak bit and good bits are not weak.

Furthermore, error-correction capabilities can be extended by implementing different ECCs on different memory levels. This approach is called *hierarchical coding*. In [37] a technique based on this idea is implemented for the NOR flash memory. A first code with low correction and extended detection capabilities is used at the word level to detect errors. When the number of errors in a word is too high, a second code with higher error correction capability is used at the page level. One drawback of this solution is the requirement for reading a whole page in case of a large number of errors occurring in one word.

In [105] a similar technique used for DRAMs is discussed. In this paper the first code used at the word level provides only error detection while the second code used for correcting data is stored in another part of the memory. Only additional word is read in case of detected errors.

Another approach presented in [2] combines BCH and Hamming codes in hierarchical manner to protect the NOR flash memory against multiple-bit errors. In [17] an adaptive-rate error correction scheme can be found. Based on BCH codes, a variable error protection level can be provided depending on the required reliability level.

Currently, latency-constrained memories (such as e.g. DRAM or NOR flash) use simple parity codes which are usually able to correct single-bit errors. But due to technology and scaling related problems, emerging memories will require codes or solutions which will be able to correct greater number of errors. Unfortunately, the hardware overhead imposed by the parallel or combinatorial implementation of multi-bit error-correcting codes (like BCH, RS) may be unacceptable for most designs. Therefore, in order to mitigate the mentioned problem, system designers should focus more deeply on memory fault types, their distribution, and the structure of the memory system. This information enables a possibility that multi-bit ECCs would not be needed. Instead, well-designed, simple parity codes with fast decoding and encoding procedures adjusted for providing stronger error protection for the most vulnerable memory areas would be sufficient.

For memories where the majority of errors are caused by radiation or electromagnetic

contamination usage of multi-bit error correcting codes is justified. However, this does not necessarily have to be the case for emerging non-volatile technologies which show immunity to radiation induced errors. Instead, the main purpose of techniques based on ECCs has to be focused on random, hard-error repair with the ability to correct errors occurring in the peripheral circuitry.

1.2.4 Comprehensive approaches

The idea of combining different fault-tolerant mechanisms is not a new one and is often utilized to improve the reliability of semiconductor memories. As an example, the ECC and the BISR can be successfully merged into a consistent system in order manage memory hard and soft errors. That is, because of the memory organization, multiple-bit hard errors occurring within a bit line can be transformed into single-bit errors in multiple bytes or words. While those errors can be corrected using ECC-based technique, multi-bit hard errors affecting a word line are harder or even impossible to repair by the ECC. For that reason, in complex designs both ECCs and redundancy repair techniques are often employed.

The ECC can be also used as a trigger for redundancy repair procedures. In [96] the ECC is used to distinguish between soft and hard errors. If an error is detected by the ECC circuit, the corrected data is written back and then read again to check whether the error remains. If the data is still incorrect, the write-back-read steps are repeated. If data is corrected after a predefined time, the detected error is considered as a soft error, otherwise, it is considered as a hard error. Once the hard error is detected, the system can enter a hard-repair state immediately or when memory is idle. In the hard-repair phase, the redundancy analysis algorithm is executed to replace the faulty memory word with a redundant one.

In [36] a mix of ECC, threshold voltage analysis, and row redundancy is used for an embedded flash memory. If during the read operation uncorrectable errors are detected by the ECC, several additional read operations with different voltage levels are performed. Further, if weak memory cells are detected, their positions are used to correct the output data and the corresponding memory row is assumed as corrupted. Next, the redundancy repair is activated. During the repair procedure data from corrupted memory row is read and stored in the spare memory row assigned for replacement.

In [106] a very interesting combination of the ECC and redundancy repair called *FREE-p* is proposed. In the FREE-p approach, each 64B memory block is protected with 6-bit error-correcting and 7-bit error-detecting (6EC-7ED) BCH code. Moreover, the design of the ECC module offers variable repair capabilities. That is, the low-cost (in terms of correction latency) ECC design is employed for most blocks which have little or no errors while high-latency correction is used only for blocks with multi-bit errors. To be more specific, the authors implement ECC design which can work in three stages called: *quick-ECC*, *slow-ECC*, and *mem-ECC*. In the quick-ECC stage the 6EC-7ED BCH detects up to 7 errors but corrects only up to 2 errors. In case when in the quick-ECC 3 or more errors are detected, the slow-ECC is invoked. In the slow-ECC stage, up to 4 errors can be corrected. If in the slow-ECC more than 4 errors are detected, the mem-ECC stage

is invoked. In the mem-ECC stage up to 6 errors can be corrected. Moreover, when the mem-ECC stage is invoked, the controller identifies which errors are wear-out errors by writing and re-reading data to/from NVM (write-verify scheme). If a block has 5 or more worn-out bits, the memory controller remaps the corrupted memory block.

The remapping is achieved by embedding a remapping pointer in the corrupted memory block. As a consequence, the mapped-out block, otherwise useless, is used as a free storage for remapping information [106]. In order to indicate which blocks have been remapped, a single bit per block (called *Data/Pointer flag*) is used. Because the remapping pointer is stored in a faulty memory block, a 7-modular-redundancy code is employed for protection of the pointer. That is, the remapping pointer is replicated 7 times within the corrupted memory block. If the remapping pointer is read, 7 replicas of the pointer are compared and the majority of identical replicas are assumed as undisturbed pointers. If the pointer cannot be correctly stored then the entire page is remapped.

In the FREE-p technique the role of the operating system is to allocate pages to map failed blocks and keep track of free slots in the allocated pages [106]. Every time a block fails, the operating system will try to remap an empty slot in the remap page for the failed block. In the absence of empty slots, it allocates a fresh page from the free page list.

The main disadvantage of the FREE-p approach is that it can be used only for large, main memories. The complexity of the approach and the necessity for the support from the operating system limits its applicability.

Another benefit of combining hard- and soft-error repair mechanisms is the synergistic effect which can be achieved [94], [61], [40]. That is, by the joint action of ECC and redundancy repair a greater reliability improvement can be reached in comparison to the sum of reliability improvements obtained by separate implementations of the ECC and the redundancy repair. The synergistic effect is produced because [94]:

- the ECC is used for correcting and detecting single-bit, soft and hard errors, while
- the BISR mechanism is responsible for managing permanent faults, thus restoring the error correction capability of the ECC.

Unfortunately, in existing fault-tolerant designs the coupling between ECC and BISR is not optimal. Due to complexity of the redundancy analysis algorithms the repair procedure is performed rarely or right after manufacturing only. Since the error correction capability of the ECC cannot be further restored the synergistic effect is not achieved. Alternatively, with state-of-the-art techniques able to perform on-line redundancy repair, either changes in the memory design or special, time consuming maintenance modes are needed. While coding techniques provide universal, transparent, on-the-fly repair, BISR requires system attention and awareness.

1.3 Research objectives

Emerging NVMs through their unique features may impact system design concepts and future electronic devices. However, in order to enable their wider adoption, concerns related to their reliability have to be addressed.

As presented in previous sections, most of memory management techniques were developed for conventional semiconductor memories. Because they do not take into account special characteristics of emerging memory technologies, their applicability for upcoming memories may not be optimal. On the other hand, techniques developed specifically for emerging NVMs are focused on large digital systems and usually require support from the operating system. Moreover, they often impose changes in the internal memory structures. As a consequence, their implementation further complicates the adoption of emerging NVMs in digital systems. Moreover, novel NVM management techniques focus on emerging NVMs only and not on NVMs in general. While emerging NVMs offer promising features, it is not yet known if they will find their place in the non-volatile memory market dominated by the flash technology. As a consequence, novel management techniques should also be able to support flash memories.

Therefore, the main research objective is to provide reliability improving techniques aimed at existing and emerging, embedded non-volatile memories which will:

- be applicable in embedded systems,
- not require changes in the internal memory components,
- provide on-line repair,
- provide comprehensive management for post-production, wear-out related, and random-bit faults that occurred in the memory array, and
- be transparent for the user and have minimal impact on other components in the system, e.g., will not require user/operating system attention.

It has to be noted that memory lifetime prolongation through endurance-improving techniques is not the objective of the thesis. Although author acknowledges the importance of these techniques, the purpose of mentioned reliability-improving mechanisms is to manage memory faults and errors once they occur in the memory array.

Chapter 2

Comprehensive approach

In order to improve the reliability of existing and emerging non-volatile memories the biggest research focus was placed on simple memory repair mechanisms. It was assumed that, by developing repair mechanisms focused on specific aspects of existing and emerging NVMs, a highly modular and configurable repair system could be provided. Moreover, by combining many repair techniques into a consistent system it was expected to produce the synergistic effect. That is, the reliability improvement achieved with the consistent system should be greater than the sum of standalone implementations of repair mechanisms.

Further, to avoid performance degradation and changes in the internal components of the memory, developed techniques should be implemented in the hardware, in the memory controller. Moreover, the memory usage with and without reliability-improving techniques should be similar, i.e., support from the operating system or software should be minimal.

2.1 Memory structure

Techniques proposed in the thesis work together on different memory granularity levels. In order to fully explain their working mechanism and their cooperation, first, some basic definitions concerning memory structure will be provided.

As presented in Section 1.1 a typical RAM consists of (Fig. 2.1):

- memory matrix built from memory cells connected by word and bit lines,
- address row decoder used for selecting appropriate word line from the memory matrix,
- address column decoder used for selecting multiple bit lines from the memory matrix,
- column input/output circuitry consisting mostly of read sense amplifiers and write drivers,
- control logic for managing all mentioned components during write/read operations (not depicted in Fig. 2.1), and

- interface logic for proper communication between the memory and external world (also not depicted in Fig. 2.1).

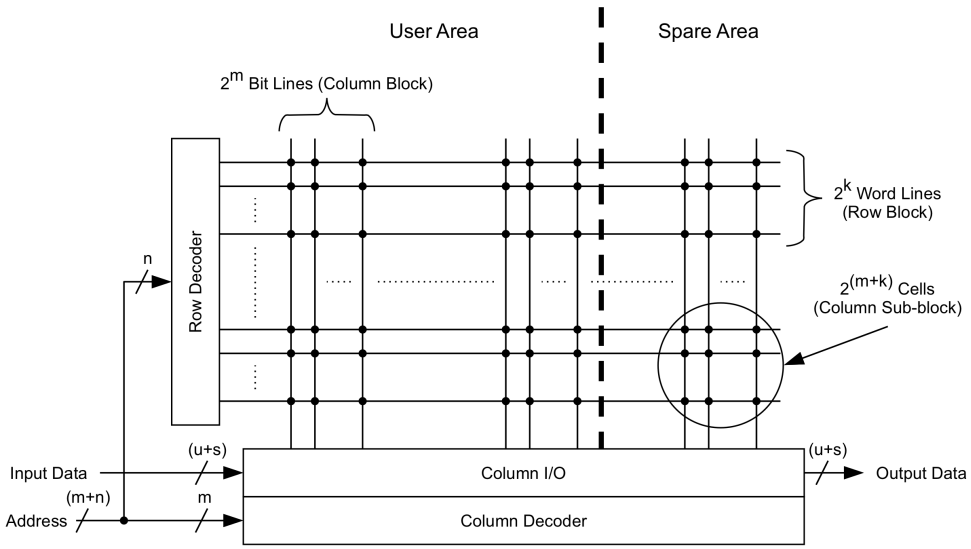


Figure 2.1: Basic memory structure with user and spare areas.

Usually, in order to achieve the best characteristics in respect to memory access time, power consumption, and reliability, word and bit lines are distributed as evenly as possible during memory design optimization phase. As a consequence, the number of bit lines is usually larger than the number of required data input/output signals. Because of that, bit lines are often grouped into column blocks where they share a single read sense amplifier and a single write driver. Each column block consists of equal number of bit lines.

In addition, memory array can be divided into row blocks (usually called *memory blocks*). The reason behind dividing memory array into row blocks can be dictated by e.g. memory specification requirements (e.g. sectors and blocks in NOR and NAND flash memories) or system-level perspective (e.g. operating system's page).

Besides area used for storing user data, often, in semiconductor memories an additional area can be found. This redundant area is typically employed for error-correcting codes and/or redundancy repair techniques. The form and organization of the redundant area depends on used fault-tolerance mechanisms. That is, in case of 1D redundancy repair the redundant area has the form of spare rows or spare columns. In case of 2D redundancy repair, the redundant area is composed of both, spare rows and columns. For the purpose of employing error-correcting codes for each memory word, the redundant area has the form of column blocks. Redundant column blocks increase the number of memory data

input/output signals.

The memory structure for which proposed techniques were developed is depicted in Figure 2.1). In Figure 2.1 memory array is divided into row blocks (each consisting of 2^k word lines) and column blocks (each consisting of 2^m bit lines). The division of the memory array into row and column blocks creates column sub-blocks (CSBs) (each consisting of $2^{(m+k)}$ memory cells). On top of that, the memory array is divided into user and spare (redundant) areas. The redundant area is composed of s column blocks. As a result, in each row block, there are u CSBs which are used for storing user data and s CSBs that will be used for the proposed techniques.

In order to simplify the description of solutions proposed in the thesis, the memory structure presented in Figure 2.1 can be transformed into a simple memory model depicted in Figure 2.2. The simple memory model represents the way how the memory is seen from the user (system-level) perspective. That is, instead of 2D array of memory cells as in Figure 2.1, the memory structure is modeled as a 1D array of memory words. The transformation was done by reducing the number of bit lines to the number of memory input/output signals. In order to preserve the memory capacity, the resulting bit lines have to be longer. As a consequence, in the simple memory model there are $(u + s)$ column blocks, each consisting of a single bit line. Each row block consist of $2^{(m+k)}$ memory words, and each CSB consists of $2^{(m+k)}$ memory cells.

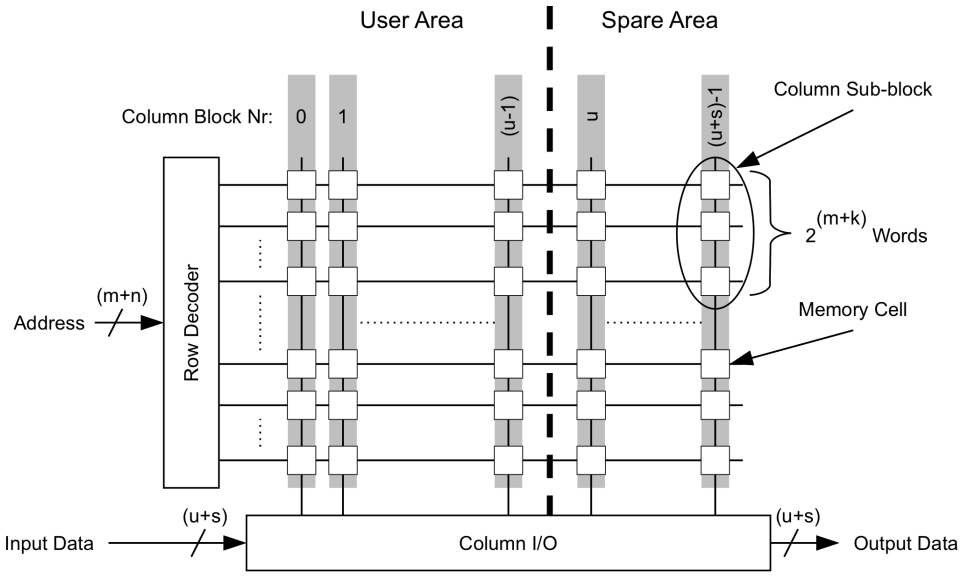


Figure 2.2: Simple memory model.

One has to note that no real bit-line reduction and bit-line length extension was per-

formed on the memory structure depicted in Figure 2.1. The simple memory model represents only a different point of view (system-level point of view) on the semiconductor memory.

2.2 IHP 8k x 44-bit NOR flash memory

Further presented reliability evaluations and implementations of the proposed fault-tolerant techniques are based on the IHP 8k x 44-bit NOR flash memory (further referred as example memory). The IHP 8k x 44-bit NOR flash memory provides 8192, 44-bit memory words where first 32-bits in each word are reserved for the user and the rest of the bits (12 bits) can be used for fault-tolerant techniques. The IHP's NOR flash memory has been chosen as a base memory for further evaluations for several reasons:

- it has the required memory structure (as presented in the previous sections),
- it provides the redundant area in the form of column blocks, and
- it suffers from erase-before-write characteristic.[1]

The IHP's NOR flash memory matrix consists of two memory sub-matrices (Fig. 2.3 and Fig. 2.4a). Each memory sub-matrix provides 8192, 22-bit memory words and can be individually enabled or disabled. As a consequence, the whole memory chip is able to provide 8192, 44-bit memory words.

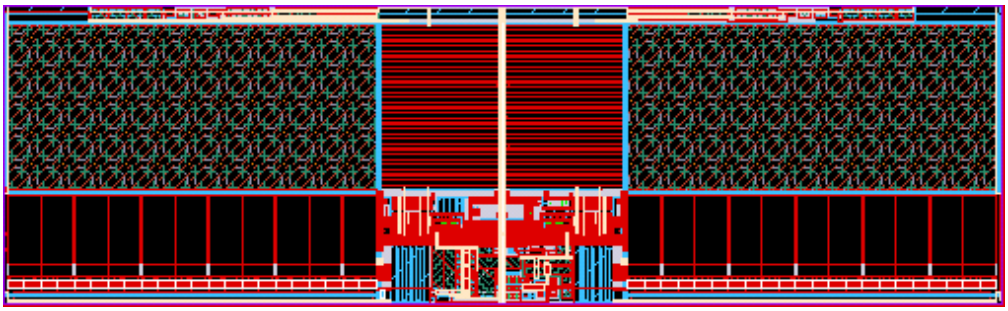
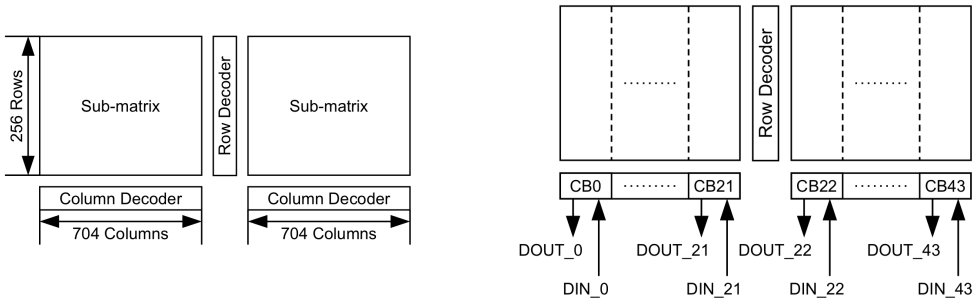


Figure 2.3: Layout of the IHP 8k x 44-bit NOR flash memory.

The memory matrix is divided into 256 rows (word lines) and 1408 columns (bit lines) (Fig. 2.4a). Bit lines are grouped into 44 column blocks where each column block consists of 32 bit lines (Fig. 2.4b). The row decoder manages 256 rows using address signals A[12], A[11], A[10], A[9], A[8], A[7], and A[6]. The column decoder manages 32 bit lines in each column block concurrently using address signals A[5], A[4], A[3], A[2], A[1], and A[0].

The communication with the chip is provided through interface signals presented in Table 2.1. Depending on the state and configuration of interface signals, the flash memory can operate in different modes (Fig. 2.5):

[1] Having flash memory as a base memory for evaluations can help verifying the applicability of proposed techniques not only for emerging NVMs but also for flash memories.



(a) (b)

Figure 2.4: Basic blocks of the IHP 8k x 44-bit NOR flash memory (a) and layout of column blocks in the memory matrix (b).

- Common write (CW) access mode provides writing of all cells in the matrix at once. This mode is used before every erase procedure. It is needed to get equal threshold voltage level of all cells before erasing (that allows excluding cell over erasing, e.g. keeping the same cells threshold voltage level after erasing). After CW only common erase mode is possible.

- Write access mode provides writing of the cells selected by address signals. After write mode any modes are possible.

- Sector write (SW) access mode provides writing of all cells in the sector at once. It is needed to get equal threshold voltage level of all cells before erasing. Size of the sector can be changed in accordance to the address table (Tab. 2.3). After SW only sector erase mode is possible.

- Common erase (CEr) access mode provides erasing of all cells in the matrix at once. This mode is used after each CW procedure. Endurance screening is additional application of this mode (together with CW mode).[2] After CEr any modes are possible.

- Sector erase (SEr) access mode provides erasing of all cells in the sector at once. Size of the sector can be changed in accordance to the address table (Tab. 2.3). This mode is used after each SW procedure. Endurance screening is additional application of this mode (together with SW mode). After SEr any modes are possible.

[2]In the CW mode all memory cells are written at once. In the CEr mode all memory cells are erased at once. As a consequence, during CW and CEr modes, the state of each memory cell will change at lest once (depending on the state of the cell before CW and CEr modes). Therefore, CW and CEr modes can be used to evaluate the endurance of memory cells.

- Read access mode provides reading of cells selected by address signals. After read mode any modes are possible.

- Standby mode switches-off the memory array. In the standby mode the memory chip requires minimal power supply in respect to other modes. After standby mode any modes are possible.

Table 2.1: Flash memory interface signals.

Name	Type	Description	Active
A[12:0]	Input	Address	High
DIN[43:0]	Input	Data in	High
DOUT[43:0]	Output	Data out	High
Read	Input	Read signal	High
Write	Input	Write signal	High
Erase	Input	Erase signal	High
CWE	Input	Common write enable	High
nCE1	Input	Chip select for the 1^{st} sub-matrix	Low
nCE2	Input	Chip select for the 2^{nd} sub-matrix	Low

Table 2.2: Flash timing.

Mode	Unit	Time
CW	μs	500
CEr	ms	50
SW	μs	50
SEr	ms	50
PreWrite	μs	300
Write	μs	100
PreRead	ns	100
Read	ns	50

In the IHP's flash memory writing and erasing a memory cell means storing logic '1' and logic '0' in the cell, respectively. Because it is a flash memory technology, once a memory cell is written (logic '1') it cannot be individually erased. Instead, the whole memory or the sector where the memory cell is located has to be erased. IHP's flash memory has the unique feature that allows changing the sector size during normal memory operation. That is, depending on the configuration of the address signals provided during SW and SEr modes (Tab. 2.3), the sector size can vary from a single memory row to the whole memory array.

Table 2.3: Flash memory address configuration for different sector sizes.

# of Sectors	# of Rows per Sector	Address Signals												
		Sector #								Sector Size				
		A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
256	1									0	0	X	X	X
128	2								X	0	1	0	0	0
64	4							X	X	0	1	0	0	1
32	8						X	X	X	0	1	0	1	0
16	16					X	X	X	X	0	1	0	1	1
8	32				X	X	X	X	X	0	1	1	0	0
4	64			X	X	X	X	X	X	0	1	1	0	1
2	128		X	X	X	X	X	X	X	0	1	1	1	0
1	256	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 2.5: Signal waveforms of IHP 8k x 44-bit NOR Flash memory in basic modes of operation.

Secure[3] timings required for different flash memory modes are provided in Table 2.2. The PreWrite mode is a first write mode which occurs after CEr, SEr, PreRead, read, and standby modes. The PreRead mode is a first read mode which occurs after CW, CEr, SW, SEr, PreWrite, write, and standby modes.

2.3 On-line redundancy repair

The proposed redundancy repair techniques are aimed at post-production and endurance-related hard faults that occur in the memory array. They were developed to improve the reliability and facilitate the management of existing and emerging non-volatile memories. Moreover, the main focus behind their creation was put on providing on-line memory repair which could be performed during normal memory operation.

The underlying repair mechanism of proposed techniques is based on replacing corrupted[4] column sub-blocks (CCSBs) with spare ones in each row block separately.

The main reasons for which the block-based redundancy was chosen is that:

- small block/bit based allocation can be more effective for replacing defective memory areas than techniques based on row/column replacement [71], [43], [27],
- no complex algorithms are required for redundancy analysis as in 2D redundancy repair techniques,
- most NVMs require charge pump circuits which are very vulnerable to radiation induced failures [32], and when defective, can influence the consistency of whole data blocks during memory write operations,
- most NVMs are based on complex storage mechanisms and require complex write circuitries, which, when corrupted, can also influence the validity of stored data (e.g., in a whole column block), and
- due to special fault mechanisms (e.g., resistance drift in PCRAMs) and influence of external factors (e.g., ambient temperature, magnetic field) faults in emerging NVMs are expected to manifest as corrupted memory areas rather than a single, random-bit errors.

In order to detect endurance and post-production faults in the memory array, the proposed redundancy repair techniques utilize the write-verify scheme where an additional read operation is performed after every memory write operation. Due to its simplicity, the write-verify scheme is commonly used in memory on-line repair techniques (e.g. ECP [81], SAFER [84], FREE-p [106]) for detecting new faults and in multi-level cell memories for achieving multi-level programming. Moreover, the write-verify scheme provides

[3]Timings which were used during flash memory tests to verify the correctness of memory operations. Provided timings are not optimal.

[4]A corrupted column sub-block is defined as a column sub-block where at least one memory cell is defective.

continuous on-line screening for new faults in the memory array. The drawback is that additional memory read operation is added to each memory write operation. Furthermore, after the read operation is performed, the additional logic is required to compare input and output data. The additional logic also introduces some delay. As a result, the implementation of the write-verify scheme have negative impact on memory performance. This problem is even more severe in situations where the memory requires longer times for first write and read operations e.g. like in IHP 8k x 44-bit NOR flash memory. As a consolation, in existing and emerging NVMs the write operation is typically orders of magnitude longer than read operation (e.g., 50-150 ns for read in contrast to 100-400 μs for write as in IHP NOR flash memory). Moreover, the time required for memory write operation is not as important as time required for read operation.

In the proposed redundancy-repair techniques, when after the memory write operation the verify procedure reports new errors, CCSBs have to be exchanged with spare ones. In order to do so, positions of CCSBs have to be stored in special pointers - error position pointers[5] (EPPs) - and appropriate spare CSBs have to be assigned for replacement.

There are two ways how positions of CCSBs can be stored. In the first way, the CSBs are indexed starting from 0. Next, when CCSB is detected, its position is stored in a EPP and enable bit for that pointer is set. In the second way, the CSBs are indexed starting from 1. Further, the position of detected CCSB is stored in the EPP (similarly to the first way) without the need for the enable bit. EPP equal to 0 defines that the pointer is not enabled and any other value than 0 defines that the pointer is enabled.

Although the way how the position of the CCSB is stored may seem insignificant, it has the influence on the required additional memory area. If the number of CSBs is a power of two then both ways require the same amount of bits to store the position of CCSB (Tab. 2.4). But if the number of CSBs is not a power of two, then the second way is more efficient. That is, using the second way more position can be stored in a single EPP (Tab. 2.5). The drawback of the second way is that more hardware is required in order to check if the EPP is enabled or not. Moreover, the position stored in the EPP may require additional calculations (e.g., adding and/or subtracting 1) when further processed. Nonetheless, because the second way can store more positions it will be used for storing the positions of CCSBs.

As a consequence, for each memory block n, $\lceil log_2(u+1) \rceil$-bit EPPs will be used for storing positions of up-to n CCSBs located in the user area. Further, an n-bit vector - corrupted spare position (CSP) vector - will be required for marking spare positions which are not eligible for replacement. For the sake of clarity, EPPs and CSP will be referred as configuration data or meta-data.

Once the CCSB is detected during normal memory operation, its position is stored in the first, unused EPP and an appropriate spare position (SP) is selected for replacement. The index of the EPP defines which SP will be used for replacement, i.e., first EPP defines first SP, second EPP defines second SP used for replacement, etc. In the word-level repair each SP consists of two CSBs (as depicted in Fig. 2.6) while in the block-level repair each SP consists of only one CSB (not depicted in Fig. 2.6).

[5]Similarly to error-correcting pointers (ECPs) as in [81].

Table 2.4: Number of bits required for EPPs.

# of CSBs (c)	# of Bits for EPP	
	1^{st} Way $\log_2(c)+1$	2^{nd} Way $\lceil \log_2(c+1) \rceil$
2	2	2
4	3	3
8	4	4
16	5	5
32	6	6
64	7	7
128	8	8
256	9	9
512	10	10

Table 2.5: Capacity of EPPs.

# of EPP Bits	# of Positions	
	1^{st} Way	2^{nd} Way
2	2	3
3	4	7
4	8	15
5	16	31
6	32	63
7	64	127
8	128	255
9	256	511
10	512	1023

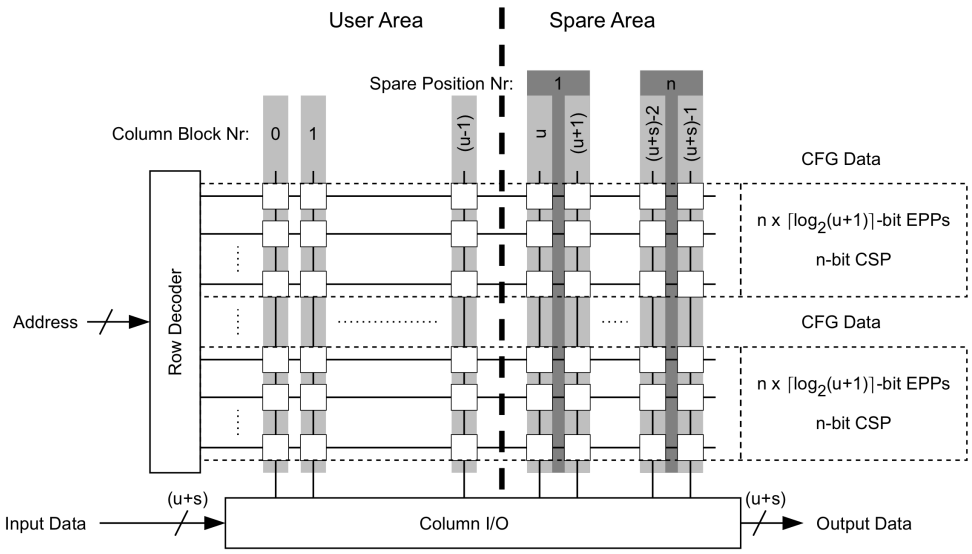


Figure 2.6: Layout of the memory array with required configuration data.

Because the configuration data needs to be accessed every read/write memory operation, it should be stored in the memory with fast access times (e.g. register file or SRAM). Furthermore, it should be protected against errors and situations when power connected to the system is lost. More information on the subject of protecting the configuration data is presented in Section 4.1. Further in the thesis it is assumed that the configuration data is stored in an error-free, non-volatile memory with fast access times.

Further, detailed description concerning word- and block-level repairs is provided.

2.3.1 Word-level repair

The working mechanism of the word-level repair will be described using a simple example depicted in Figure 2.7. Figure 2.7a shows a single memory block where the user area is composed of four 8-bit memory words and the spare area consists of two SPs. In the word-level repair each SP consists of two CSBs. As a result, in each memory word two spare bits belong to a single SP. The first bit is used as a replacement bit while the second bit indicates if the replacement bit is active. Moreover, since there are two SPs which will be used for replacement, two 4-bit EPPs and one 2-bit CSP vector are also needed. Although the memory block can store any data, for simplicity, in the presented example all memory words and meta bits are set to '0'.

After a memory write operation to the first word in the memory block (Fig. 2.7b), the verify procedure is automatically initialized to check the validity of stored data. That is, the input data is xor-ed with the output data to determine the difference vector. If the difference vector does not contain any '1' then the stored data is assumed valid. In case when the difference vector contains '1' on any position then it is compared with the vector of known faults. The known-faults vector contains '1' on positions indicated by active EPPs and the CSP vector. If the difference vector and known-faults vector have '1' on the same positions then the corresponding CCSB is assumed as already taken care of. If on the same positions only the difference vector has '1' then the corresponding CSB is assumed as new CCSB.

In Figure 2.7b both EPPs are inactive, therefore error which occurred in the 4^{th} CSB is assumed a new error and 4^{th} CSB is considered as corrupted. Further, in the error evaluation phase a single error is selected from all occurred new errors and its position is stored in the first unused EPP. Next, the value of the input data bit pointed by the first EPP is stored in the first bit of the first SP. From now on, when a write operation will be performed on a word in the memory block, the first SP in that word will be automatically activated and filled with the value of the input data bit pointed by the first EPP. Note, that although there are no errors after the write procedure to the second word in Figure 2.7c, the first SP in that word is activated and filled with the appropriate value.

In the memory block where the word-level repair is implemented, after the memory read operation, values from active SPs are automatically placed in the memory output word at positions pointed by active EPPs. If one would like to read the first word from the memory block (Fig. 2.7b), the value from the first SP - '1' - will be used instead of the value of the corrupted bit located in 4^{th} CSB. Furthermore, if the 3^{rd} word would have been read from the memory block (Fig. 2.7b) then the value of the bit located in the 4^{th} CSB block will be driven to the memory output because the first SP is not active (although the first EPP is active). This simple example illustrates the necessity of using two bits for every SP in the word-level repair. If SPs would consists of replacement bits only, after memory read operations, values from SPs would automatically replace values pointed by active EPPs. Since not all words have replacement bit properly filled, incorrect data would have been read from the memory block.

Once the SPs are assigned, replacement cells start to wear in the same way as memory cells which store user data. As a result, in addition to post-production faults, wear-out

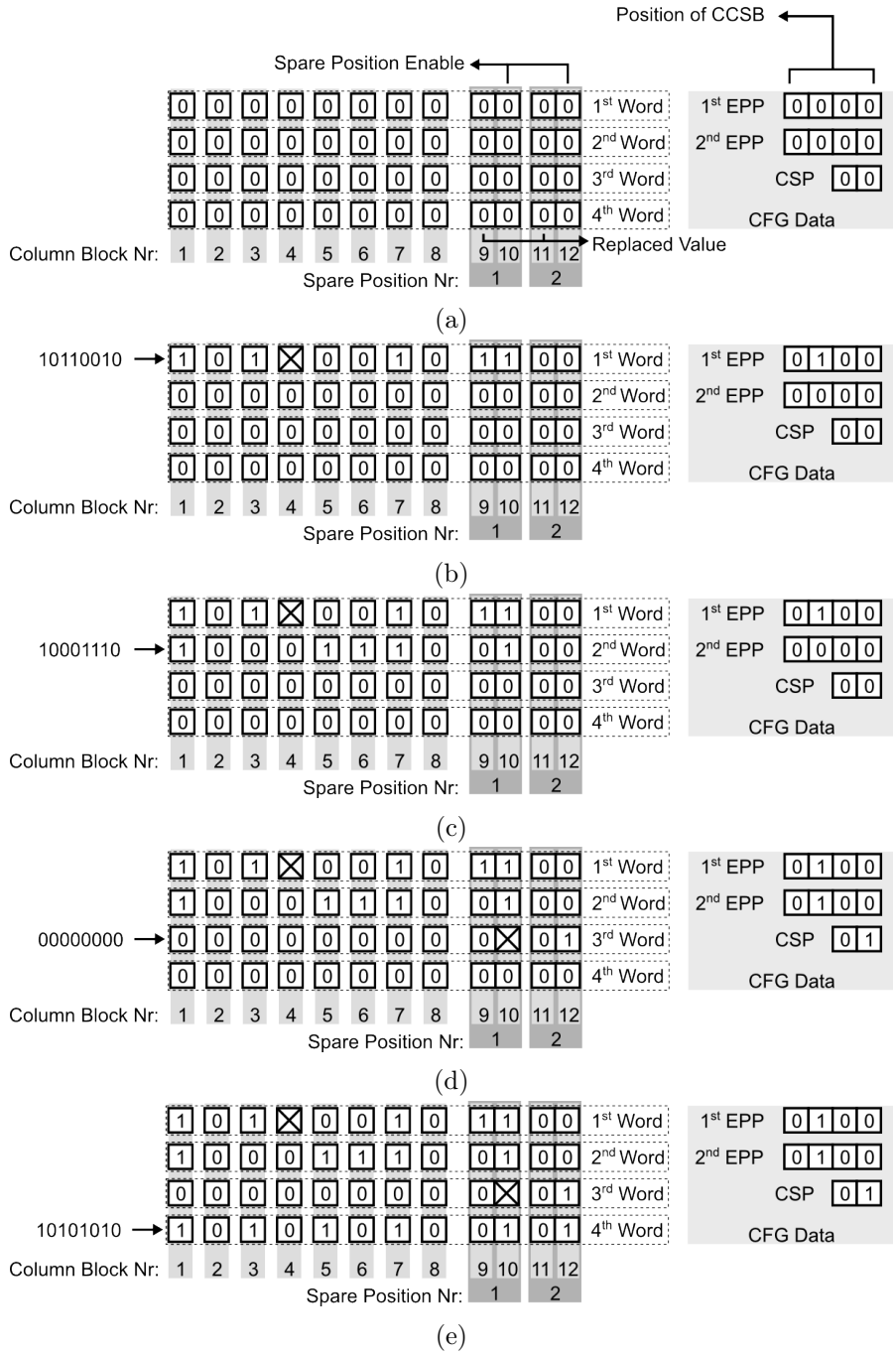


Figure 2.7: Example of the word-level repair.

caused cell failures can also occur in SPs. Example concerning error located in the SP is depicted in Figure 2.7d. After performing write operation to the 3^{rd} word in the memory block, the verify procedure detects new error in the first SP. Further, during error evaluation phase, the second EPP is filled with the position of the CCSB stored in the first EPP and the first SP is marked as corrupted. The position of the CCSB pointed by the first EPP is now considered as the position of a new error. Next, the value of the input data bit pointed by the first and second EPPs is stored in the first and second SP. From now on, two SPs manage 4^{th} CCSB (Fig. 2.7e). After the read operation, value from the second active SP will replace the value of 4^{th} data bit. In the WR, SPs with higher indexes have precedence over those with lower indexes[6]. Hence, in the example, the value from the second SP will override the value from the first SP.

In case when the second SP is not active in a word, the value of the first SP will be used for replacing the value of a bit located in the CCSB during memory read operation. If one would like to read the first word from the memory block (Fig. 2.7e) the value from the first SP - '1' will be used for replacement because the second SP is not active.

Once all SPs are used, no more errors can be repaired in the memory block.

The flow chart presenting the complete word-level repair mechanism is depicted in Figure 2.8. After each memory write operation, the verification phase is performed. In the verification phase the word is read back from the memory and compared with the input data. Next, the output from the comparison is checked against positions of already known CCSBs. If there are no new errors then the verification phase is ended and no repair is performed. In case where there are new errors, the error evaluation phase is executed. In the error evaluation phase, first, a single error is selected. Next, it is checked whether the new error is located in unused SP. If yes then the unused SP is marked as corrupted in the CSP vector. SP marked as corrupted in the CSP vector cannot be used for a replacement. As a consequence, the corresponding EPP will also not be used. Updated CSP vector changes the information about known CCSBs. As a result, once again output from the input-output data comparison is checked against positions of already known CCSBs. If new error is located in used SP then the used SP is marked as corrupted in the CSP vector and the position stored in the corresponding EPP is considered as a position of a new error. Because of that, there is no need for the EPP to store positions of CCSBs located in the spare area. In case when there is an error in used SP then the next available SP will be used instead of corrupted SP for replacing CCSB from the user area.

Further in the error evaluation phase it is checked whether there are any unused EPPs. If yes, then the position of a new error is stored in the first, unused EPP. Next, according to active EPPs, appropriate bits in used SPs are filled and activated and updated input data is stored in the memory. Then, the whole verification and error evaluation (in case of new errors) procedures are executed until there are no new errors or there are no free EPPs left.

The word-level repair is able to perform on-line redundancy repair and does not require any complex redundancy analysis algorithms. Moreover it can be implemented even for

[6]Similarly like in the ECP-based technique [81].

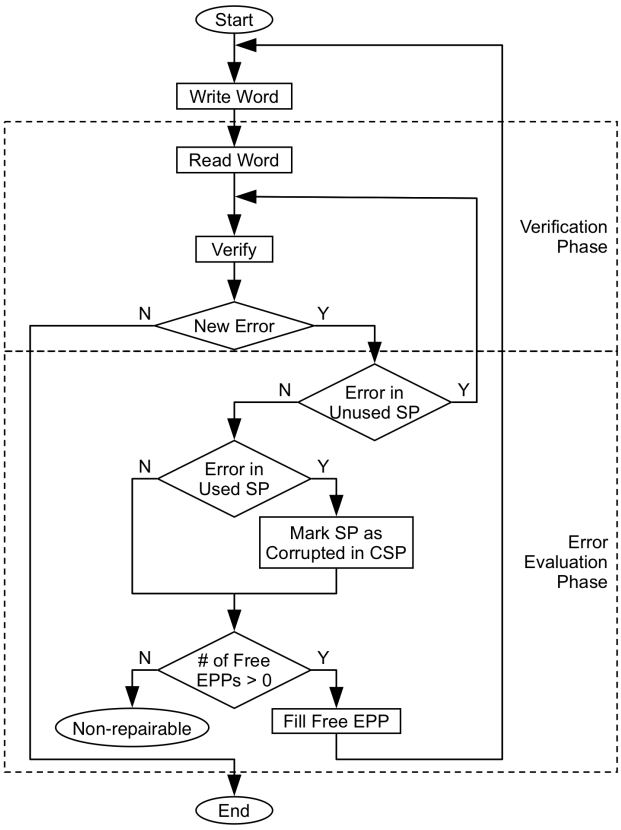


Figure 2.8: Flow chart of the word-level repair.

flash memories which suffer from erase-before-write characteristic. That is, during repair procedure only spare bits change their state (only from '0' to '1' state). The word bits located in the user area are untouched. The disadvantage of the proposed implementation is that the word-level repair can replace only one CCSB at once. As a consequence, in the worst case, c additional memory write operations and $(c+1)$ additional memory read operations are required to replace c CCSBs. It could be possible to replace more CCSBs at once but it would require more complex error evaluation procedure.

In order to evaluate the yield improvement which can be achieved with the word-level repair, first, some basic definitions will be provided. Let assume a memory chip with n memory cells. Further, let p be the probability that a single memory cell is faulty. Moreover, let assume that faults are randomly distributed in the memory array and the probability (p) that a single memory cell is faulty is independent from other memory cells being faulty or non-faulty. As a result, the raw yield[7] of the memory array can be

[7]The raw yield is defined as a yield of the memory where no reliability improving techniques were implemented.

expressed as a product of probabilities of all non-faulty memory cells [40]:

$$Y_{raw} = (1-p)^n \tag{2.1}$$

For calculating the probability of having k faulty cells in the memory array one can use the binomial distribution [40]:

$$P(k) = \binom{n}{k} \times p^k \times (1-p)^{(n-k)} \tag{2.2}$$

Where $\binom{n}{k}$ is called *binomial coefficient* and is expressed as:

$$\binom{n}{k} = \frac{n!}{(n-k)! \times k!} = \frac{n \times (n-1) \cdots (n-k+1)}{k!} \tag{2.3}$$

Because the number of memory cells (n) is usually very large, the probability that a memory cell is faulty (p) is very small, and $n \gg k$, the binomial distribution is not the most efficient tool for determining the probability of having k faulty elements in the memory array. Instead, the Poisson distribution is mostly used for rough yield analysis and a comparison of reliability improving techniques [40].

The Poisson distribution is a special case of binomial distribution and can be used for approximating large number of Bernoulli trials [40]. Let assume that the $n \to \infty$, $p \to 0$, and $n \gg k$. Further, let introduce $\lambda = n \times p$. Next, substituting $p = \frac{\lambda}{n}$ in 2.2 results in:

$$\begin{aligned} P(k) = \frac{n \times (n-1) \cdots (n-k+1)}{k!} \times p^k \times (1-p)^{(n-k)} = \\ 1 \times \left(1 - \frac{1}{n}\right) \cdots \left(1 - \frac{k+1}{n}\right) \times \frac{\lambda^k}{k!} \times \left(1 - \frac{\lambda}{n}\right)^n \times \left(1 - \frac{\lambda}{n}\right)^{-k} \to \\ \frac{\lambda^k}{k!} \times \left(1 - \frac{\lambda}{n}\right)^n = \frac{\lambda^k \times \exp(-\lambda)}{k!} \end{aligned} \tag{2.4}$$

Formula derived in 2.4 defines the Poisson distribution [40]. The average $\bar{k}$ is expressed as:

$$\bar{k} = \sum_{k=0}^{\infty} k \times P(k) = \exp(-\lambda) \times \sum_{k=1}^{\infty} \frac{\lambda^k}{(k-1)!} = \lambda \tag{2.5}$$

The Poisson distribution is characterized by only one parameter λ which is equal to the average number of faults in the memory array. As a consequence, the raw yield of a memory array can be characterized by:

$$P(0) = \exp(-\lambda) \tag{2.6}$$

The use of Poisson distribution can facilitate memory yield estimations concerning the average number of faults which can occur in the memory array [40]. For example, let

assume that X memory chips were fabricated and after memory tests it was discovered that on average one memory fault can occur in the memory array. Then, substituting $\lambda = 1$ in 2.6 we get:

$$P(0) = \exp(-1) \approx 0.3679 \tag{2.7}$$

which defines that on X memory chips, on average, we should expect $\approx 0.3679 \times X$ memory chips with fault-free memory arrays. The raw yield of the memory array in respect to the average number of memory array faults is depicted in Figure 2.9.

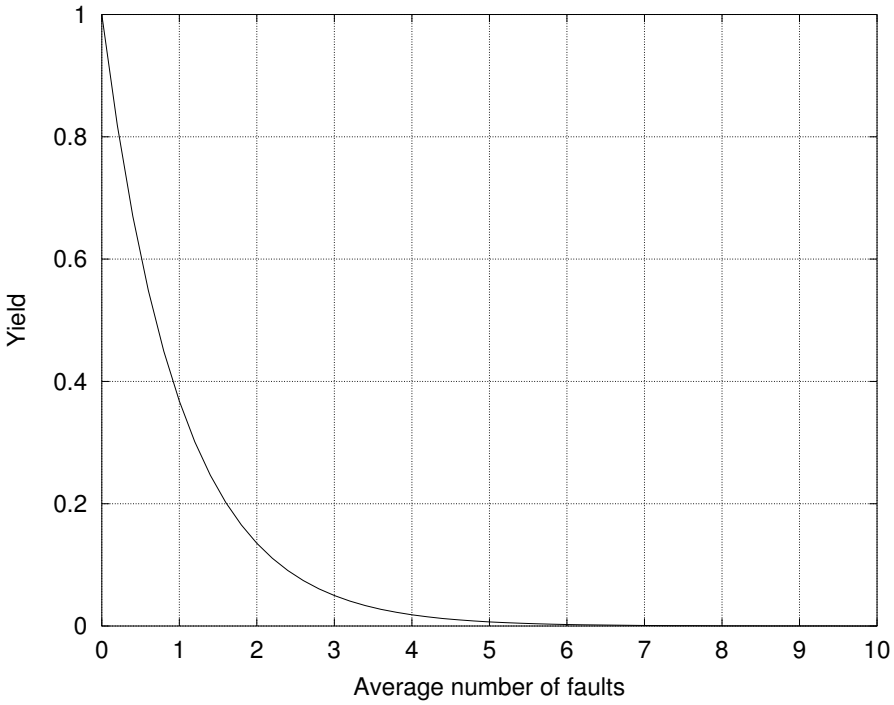


Figure 2.9: Raw yield of the memory array in respect to the average number of random-bit faults.

Further reliability calculations concerning word-level repair will be based on the IHP 8k x 44-bit NOR flash memory presented in Section 2.2. The word-level repair is based on exchanging corrupted column sub-blocks with spare ones for each memory block separately. In the example memory, there are 44 CSBs in each memory block and b memory blocks in the memory array. The yield of a single CSB can be expressed by:

$$Y_{csb} = \exp\left(-\frac{\lambda}{44 \times b}\right) \tag{2.8}$$

and the raw yield of the whole memory by:

$$Y_{raw} = Y_{csb}^{(44 \times b)} \tag{2.9}$$

But from the user point of view, if no reliability-improving techniques are implemented, faults occurring in the spare area are not significant. As a consequence, the yield of the user area can be calculated with:

$$Y_{raw\_nr} = Y_{csb}^{(32\times b)} \tag{2.10}$$

In the example memory there are 12 spare bits in each memory word which can be used for the word-level repair. Because SPs in word-level repair require 2 CSBs, up to 6 SPs can be implemented in each memory block.

Let start with a simple example where the whole memory array is treated as a single memory block ($b = 1$) and only one SP is used for replacement. As a consequence, each memory word consists of 44 bits but only 34 bits are used (32 bits in the user area and 2 bits in the spare area). The yield of the memory block can be calculated with:

$$Y_b = Y_{csb}^{34} + \binom{34}{1} \times Y_{csb}^{33} \times (1 - Y_{csb}) \tag{2.11}$$

That is, the memory block can be fault free (the Y_{csb}^{34} part) or it can have a single CCSB (the $34 \times Y_{csb}^{33} \times (1 - Y_{csb})$ part) which will be replaced with SP.

If the WR able to replace two CCSBs is implemented, then the yield of a memory block can be expressed by:

$$Y_b = Y_{csb}^{36} + \binom{36}{1} \times Y_{csb}^{35} \times (1 - Y_{csb}) + \binom{36}{2} \times Y_{csb}^{34} \times (1 - Y_{csb})^2 \tag{2.12}$$

Where, the memory block can be fault free (the Y_{csb}^{36} part), it can have a single CCSB which will be replaced with a single SP (the $\binom{36}{1} \times Y_{csb}^{35} \times (1 - Y_{csb})$ part), or it can have two CCSBs which can be replaced with two SPs (the $\binom{36}{2} \times Y_{csb}^{34} \times (1 - Y_{csb})^2$ part).

Similar approach can be used to determine the yield of the memory with WR implementations able to replace 3, 4, 5, and 6 CCSBs. Results concerning different implementations of the WR in the example memory are depicted in Figure 2.10. In Figure 2.10, Y_{raw} defines yield of the whole memory array (8192 word x 44-bits), Y_{raw_nr} defines the yield of the user part of the memory array (8192 words x 32-bits) and WRx defines WR capable of replacing x CCSBs in a whole memory (the number of memory blocks in the memory $b = 1$). Further, straight lines on the plot represent results acquired from yield formulas and "+" signs represent results acquired from computer simulations.

For computer simulations the memory array structure of the IHP 8k x 44-bit NOR flash memory and the implemented WRx technique were modeled in the C language. Next, for each modeled memory cell a probability of failure based on the average number of random-bit faults in the memory array was assigned. Further, for each memory cell a random number from $(0, 1)$ was generated. If the random number generated for a cell was lower than the probability of cell's failure then that cell was marked as corrupted. Next, memory array with corrupted bit faults was checked whether it can be corrected with WRx or not. If the WRx was able to correct all faults then the $repair\_success$ counter

was incremented. For each number of random-bit faults in the memory array 10000 computer simulations were performed with different fault distributions in the memory array and the resulting memory yield ($repair\_success/10000$) was reported.

As expected, the implementation of the WR capable of replacing more CCSBs provides better yield improvement (Fig. 2.10). Furthermore, as more CCSBs can be replaced, the difference in yield improvement between different implementations of the WR diminishes. For example, note the differences in yield improvement achieved with $WR1$ and $WR2$, and $WR5$ and $WR6$. This is caused by random-bit faults occurring in the SPs. That is, the bigger is the spare area, the higher is the probability that CSBs assigned for replacement will be corrupted.

In Figures 2.12, 2.13, and 2.14, the results concerning yield improvements achieved with $WR6$ implemented for different number of blocks in the memory are presented[8]. Of course the bigger is the number of memory blocks, the better is the resulting yield improvement. Since the $WR6$ is implemented for each memory block independently, the probability of finding more than 6 CCSBs in each memory block diminishes (Fig. 2.11).

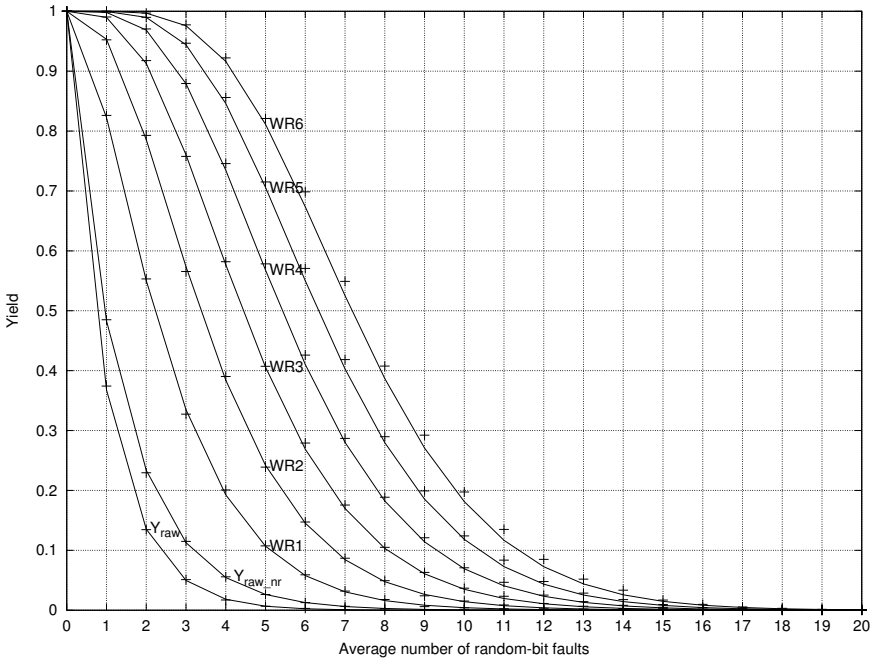


Figure 2.10: Yield of the memory array with different implementations of the WR (b=1).

Further, as the number of memory blocks increases, the area required for storing the configuration data also increases. For a single block where WR6 is implemented, 42 bits are required to store 6, 6-bit EPPs, and one 6-bit CSP vector. The total area required for the WR6 implemented for the example memory ($b = 1$) including the area required for SPs is $42 + 8192\,words \times 6\,EPPs \times 2\,CSBs\,for\,each\,SP = 98346$ bits which is

[8]Due to long simulation times, only results calculated with yield formulas are reported.

$\approx 27,3\%$ of the example memory area. The area required for different number of blocks is presented in Table 2.6.

Table 2.6: Area required for the WR6 implemented for different number of memory blocks.

# of Blocks	Area Overhead (Bits)		Ratio (%)	
	CFG Data	SPs	CFG Data to SPs	(CFG Data + SPs) to Memory Area
1	42	98304	0,043	27,284
2	84	98304	0,085	27,296
4	168	98304	0,171	27,319
8	336	98304	0,342	27,366
16	672	98304	0,684	27,459
32	1344	98304	1,367	27,646
64	2688	98304	2,734	28,018
128	5376	98304	5,469	28,764
256	10752	98304	10,938	30,256
512	21504	98304	21,875	33,239
1024	43008	98304	43,750	39,205
2048	86016	98304	87,500	51,136
4096	172032	98304	175,000	75,000
8192	344064	98304	350,000	122,727

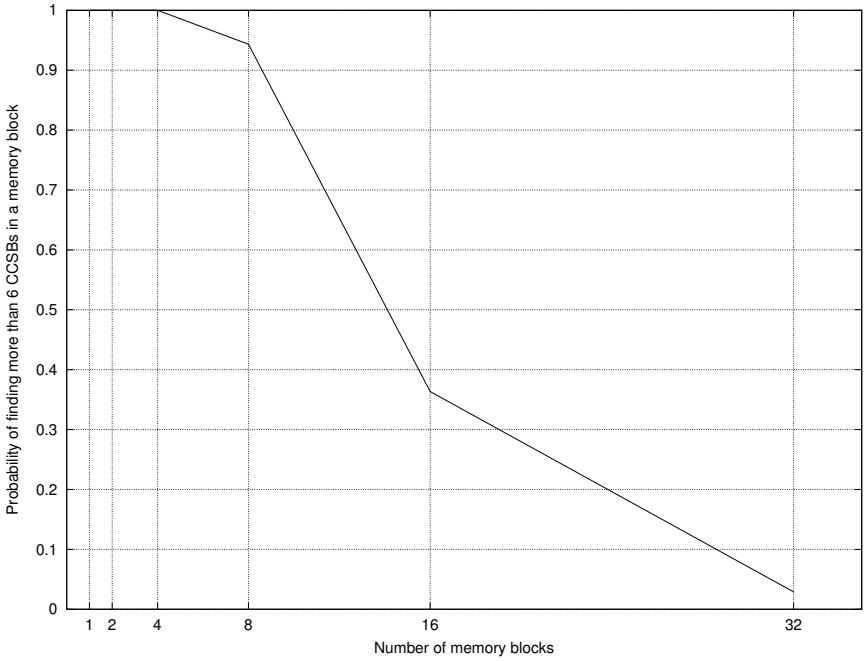


Figure 2.11: Probability of finding more than 6 CCSBs in a memory block for different number of memory blocks (1-32) and 100 random-bit faults.

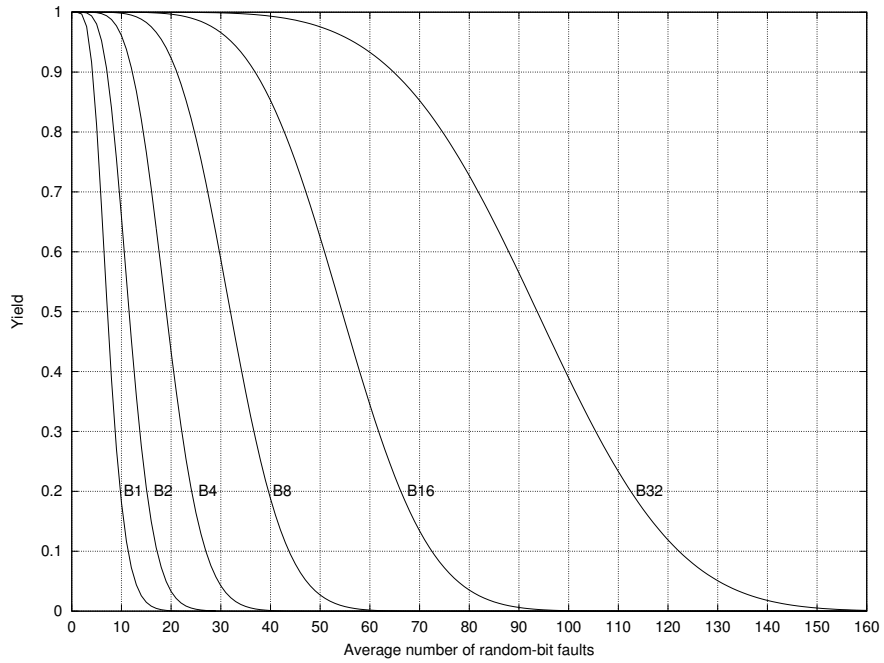


Figure 2.12: Memory array yield improvement achieved with the WR6 technique implemented for different number of memory blocks (1-32).

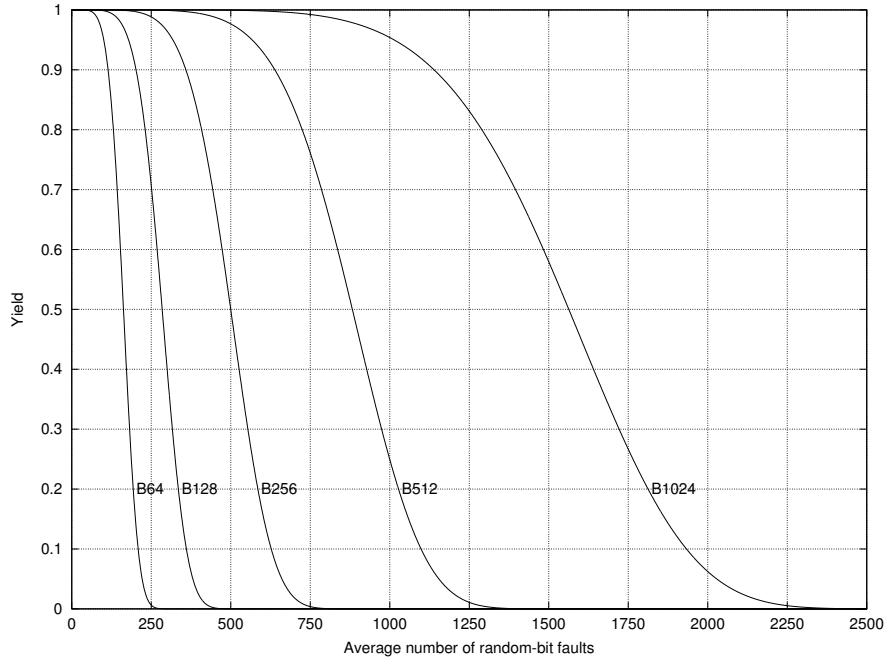


Figure 2.13: Memory array yield improvement achieved with the WR6 technique implemented for different number of memory blocks (64-1024).

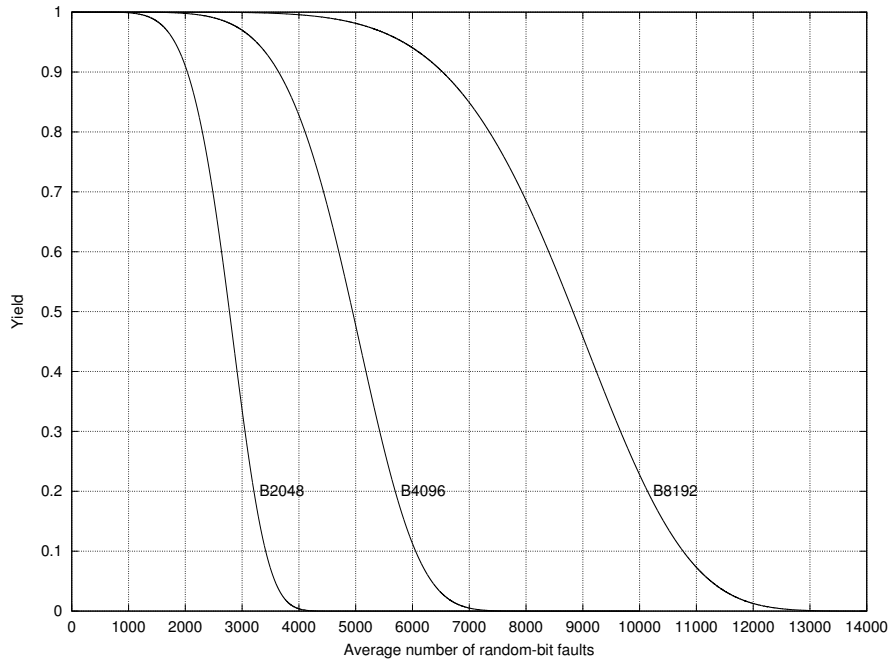


Figure 2.14: Memory array yield improvement achieved with the WR6 technique implemented for different number of memory blocks (2048-8192).

2.3.2 Block-level repair

The word-level repair requires two CSBs for a single SP because not all words in the memory block have appropriate values stored in the SP. If after detecting a new error in the memory block, all memory words would have correct values stored in the assigned SP then the number of required CSBs for a single SP could be reduced. Based on this observation a block-level repair (BR) mechanism is proposed. The principles of the BR are similar to that of the WR with two differences. First one is that each SP consists of a single CSB. The second difference is that after occurrence of a new error, a replacement bit located in the assigned SP have to be updated in every word in the memory block.

In Figure 2.15 a similar example as presented for the WR is depicted. Figure 2.15a shows a single memory block where the user area is composed of four 8-bit memory words and the spare area consists of two SPs. In the BR each SP consists of a single CSB which is used to provide replacement bits only. Further, since there are two SPs, two 4-bit EPPs and one 2-bit CSP vector are also needed. For simplicity all memory words and meta bits are filled with '0'.

In Figure 2.15b after performing memory write operation to the first word, verify procedure detects new error in the 4^{th} CSB. Next, in the error evaluation phase the position of the 4^{th} CSB is stored in the first, unused EPP and first SP is assigned for replacement. Next, the value of the input data bit pointed by the first EPP is stored in the replacement bit belonging to the assigned SP. Once the spare area of the corrupted

word is updated, next word in the memory block is read in order to acquire the value of a bit located in the 4^{th} CSB. Further, the acquired value is stored in the replacement bit belonging to the assigned SP in the second word. The same procedure is continued until all words in the memory block have appropriate values stored in the replacement bit belonging to the assigned SP. Similarly like in the WR, the BR can replace only one CSB at a time. Therefore, when during the BR procedure new error is detected it is ignored and repair procedure is continued. After the BR procedure all words in the memory have their first SP filled with the appropriate value. Similarly as in the WR, from now on, the first SP for each word will be automatically filled during word update (Fig. 2.15c).

In situation when error affects used SP (Fig. 2.15d) the affected SP is marked as corrupted in the CSP vector and the position of the CCSB stored in the first EPP is considered as the position of a new error. Further, in the error evaluation phase, second EPP is used to store position of a new error and a standard BR procedure is performed in order to update replacement bit belonging to the second SP in all words in the memory block.

As in the WR, SPs with higher indexes have precedence over SPs with lower indexes. As a result, if one would like to read the 4^{th} word (Figure 2.15e), value from the second SP will be used for replacing the value of the bit located in the 4^{th} CCSB.

The flow chart presenting a complete block-level repair mechanism is depicted in Figure 2.16. The verification and error-evaluation phases in the BR are the same as in the WR. The difference lies in the repair procedure. In the WR the repair procedure consists of rewriting the corrupted memory word in order to update the spare area in that word. In the BR the spare area in all words in the memory block have to be rewritten.

While for a single fault in a memory word one additional memory write and one memory read operation required by the WR procedure may be tolerable, memory block rewrite operation required by the BR procedure may be unacceptable during normal memory operation. Furthermore, new errors which occur during the repair procedure have to be ignored since the implementation of the BR is capable of replacing one CCSB at a time. Moreover, in the WR the repair procedure continues until all CCSBs are exchanged or there are no available SPs left. In the BR, once a single CCSB is replaced, the repair procedure ends. If there are more CCSBs then the BR procedure has to be manually restarted in order to replace all CCSBs.

As a consequence for the price of more efficient utilization of the spare area the BR suffers from limited on-line repair capabilities. That is, while WR can be easily performed during normal memory operation, the BR should be rather executed right after memory production or during special maintenance modes. The reason for that are unmanageable errors which can occur during the repair procedure and which can corrupt data stored in the memory block.

The on-line repair capabilities of the BR can be increased by making a backup of the content of the memory block before the BR procedure is executed. That is, once the new error is reported by the verify procedure the error evaluation phase should be prevented from executing. Next, the input word and content of the memory block where error occurred should be stored in the buffer or in system's RAM. Further, the same input

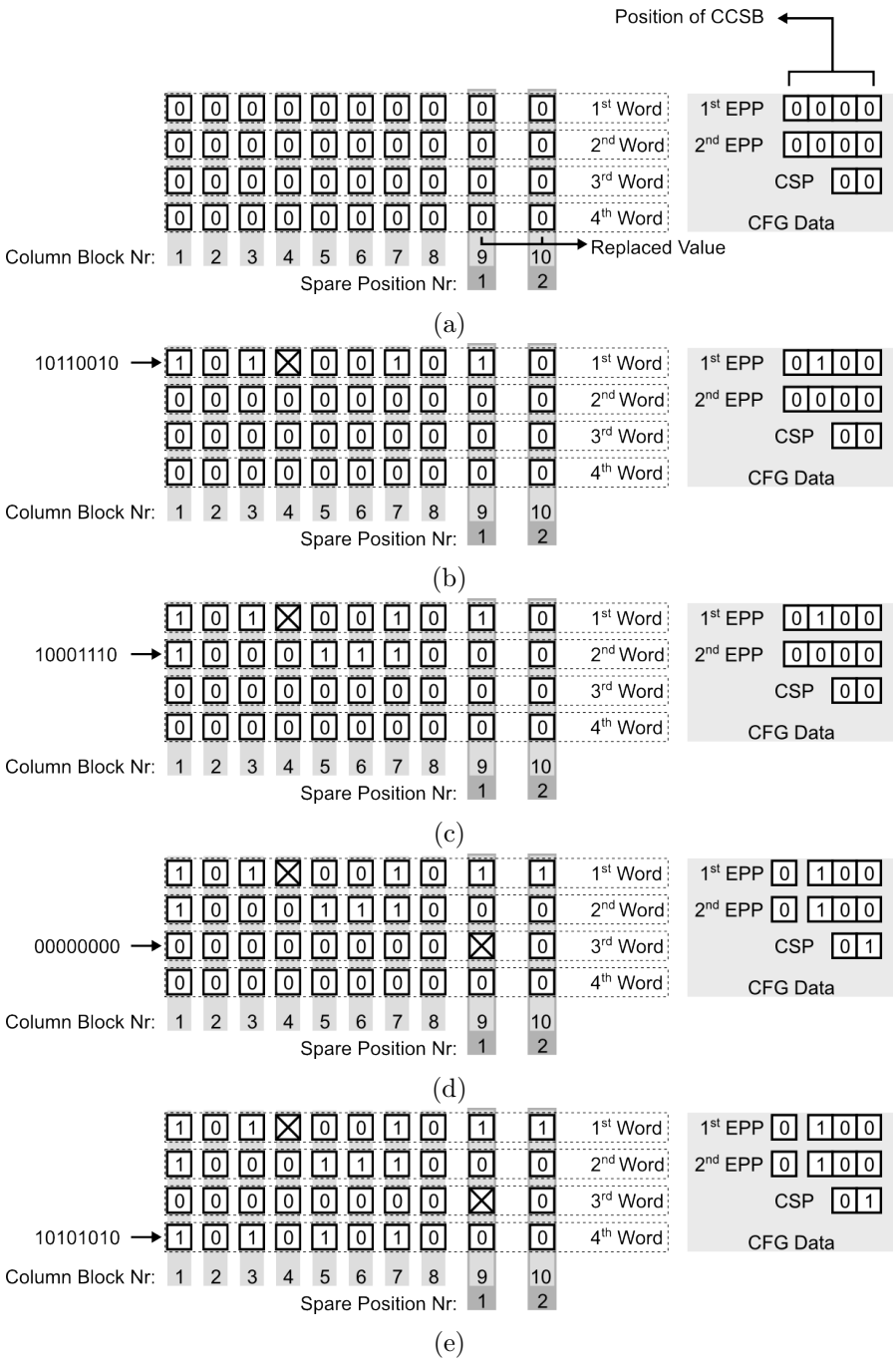


Figure 2.15: Example of the block-level repair.

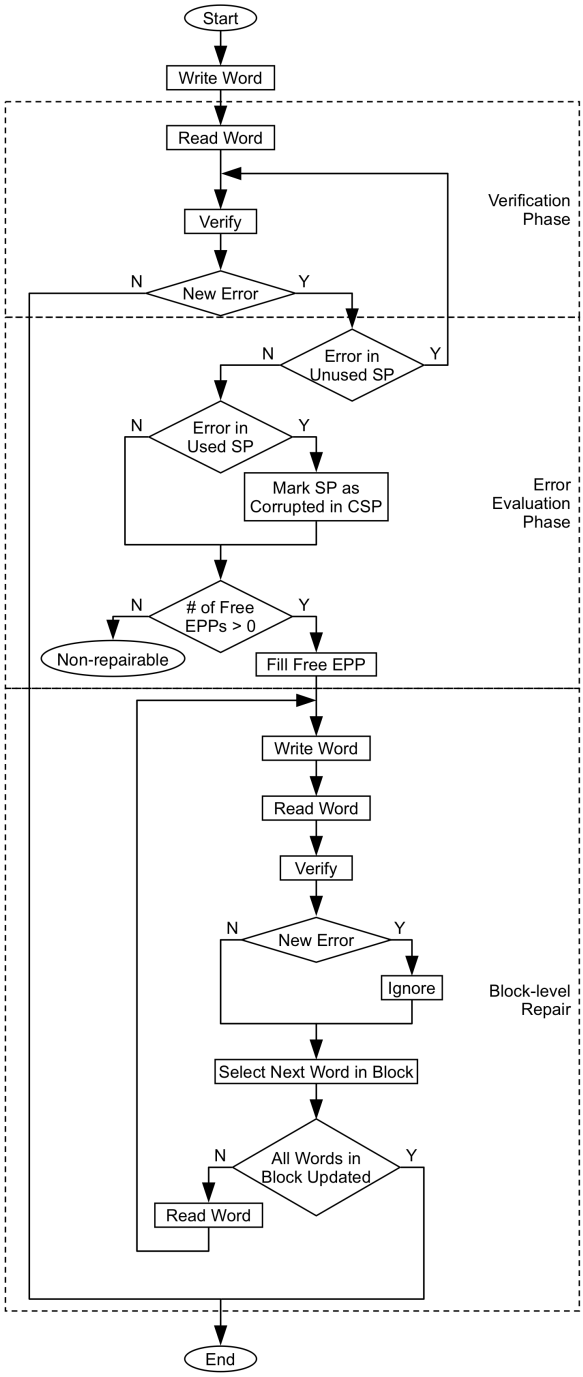


Figure 2.16: Flow chart of the block-level repair.

word should be used during memory write operation to the corrupted word. By doing this the verify procedure should report the same new error. Next the error evaluation phase, as well as BR procedure should be executed. If during the repair procedure no new errors occur then consistency of data stored in the memory block should be preserved.[9] In case when new errors occur during BR procedure, addresses of corrupted memory words should be stored. Next, after the end of the BR procedure, the memory write operation should be performed to the corrupted memory words in order to activate another BR procedures. This approach should be continued until all CCSBs are replaced with spare CSBs by the BR or there are no free SPs left. Once all CCSBs are exchanged, the original content of the memory block should be restored from the buffer or system's RAM.

Another aspect which has to be considered before the BR can be implemented in a NVM is the memory technology. Different BR implementation is required for flash memories than, e.g., for emerging NVMs. The reason is the erase-before-write characteristic of the flash memories. That is, during BR procedure it may happen that values of the corrupted bits located in unused memory words will be stored in the assigned SP. Next, when data will be stored in the unused memory word it may happen that the replacement bit belonging to the used SP is already written. For example, let assume an erased flash memory block whose a whole CSB located in the user area is corrupted (whole CSB consists of stuck-at-1 cells). Next, when the first word in the memory block will be written (e.g., with all0 pattern), the verify procedure will detect new error and BR procedure will be performed. During BR procedure, words from the memory block will be read and appropriate replacement bit in each word will be filled with the value of a bit located in the CCSB. Since the CCSB consists of stuck-at-1 cells, the replacement bit in each word will be filled with '1'. After the BR procedure when the second word in the memory block will be written (e.g., also with all0 pattern), then the verify procedure will report new error in the SP. The reason is that writing all0 pattern to the second word determines that '0' value is expected to be read from the replacement bit. But since the replacement bit located in the SP is already storing '1' and transition from '1' to '0' in flash memories can be achieved only through memory/sector erase, new error in the SP will be reported. In order to solve this problem a block erase operation is required after each BR procedure in flash memories. In memories which does not require erase-before-write the mentioned issue is not a problem and memory/block erase operation is not required.

Table 2.7 shows different implementations of the BR procedure depending on the planned use of the redundancy repair and memory technology. The maintenance mode in Table 2.7 determines situation when data preservation is not important (e.g. during post-production tests or after memory erase). The on-line mode determines normal memory operation where data stored in the memory blocks should remain valid after BR procedures.

The evaluation of the yield improvement which can be achieved with the BR will be based on the example memory model. First, let us assume that the whole memory array

[9] In order to be sure if the data stored in the block is correct, the memory block content should be compared with backed-up data.

Table 2.7: BR implementations for different types of NVMs.

	Flash Memory		Emerging NVMs	
	Block Back-up	Block Erase after BR	Block Back-up	Block Erase after BR
Maintenance Mode	No	Yes	No	No
On-line Mode	Yes	Yes	Yes	No

is treated as a single memory block ($b = 1$) and BR capable of replacing a single CCSB is implemented. As a result, from the 44-bit word only 33 bits are used. First 32 bits are reserved for the user data and a single spare bit is used as a replacement bit. The yield of such memory can be expressed by:

$$Y_b = Y_{csb}^{33} + \binom{33}{1} \times Y_{csb}^{32} \times (1 - Y_{csb}) \tag{2.13}$$

That is, the memory block can be fault free (the Y_{csb}^{33} part) or it can have a single CCSB (the $33 \times Y_{csb}^{32} \times (1 - Y_{csb})$ part) which will be replaced with the SP.

If the BR able to replace two CCSBs is implemented, then the yield of a memory block can be expressed by:

$$Y_b = Y_{csb}^{34} + \binom{34}{1} \times Y_{csb}^{33} \times (1 - Y_{csb}) + \binom{34}{2} \times Y_{csb}^{32} \times (1 - Y_{csb})^2 \tag{2.14}$$

Where, the memory block can be fault free (the Y_{csb}^{34} part), it can have a single CCSB which will be replaced with a single SP (the $\binom{34}{1} \times Y_{csb}^{33} \times (1 - Y_{csb})$ part), or it can have two CCSBs which can be replaced with two SPs (the $\binom{34}{2} \times Y_{csb}^{32} \times (1 - Y_{csb})^2$ part).

Similar approach can be used to determine the yield of the memory with BR implementations able to replace greater number of CCSBs. In the example memory the user area consists of 32 CBs and the spare area consists of 12 CBs. As a consequence, in the example memory BR able to replace up to 12 CSBs in each memory block can be implemented.

Results concerning different implementations of the BR in the example memory are depicted in Figure 2.17. In Figure 2.17, Y_{raw} defines yield of the whole memory array (8192 word x 44-bits), Y_{raw_nr} defines the yield of the user part of the memory array (8192 words x 32-bits) and BRx defines BR capable of replacing x CCSBs. Straight lines in the plot represent results acquired from yield formulas and "+" signs represent results acquired from computer simulations. Computer simulations were performed in the similar way as for the WR technique described in the previous section.

Similarly to the WR, the more CCSBs can be replaced, the better the yield improvement can be achieved. Moreover, also in case of the BR the difference in yield improvement between different implementations of the BR diminishes. This is again caused by the fact that the bigger is the spare area, the higher is the probability that SPs will be corrupted. Furthermore, since the BR requires only one CSB for a single SP, for the same number of replaced CCSBs, BR achieves better memory yield improvement

than WR (Tab. 2.8). For example, for the yield equal to 0.5, the BR6 can correct ≈ 8 random-bit faults on average while the WR6 ≈ 7 random-bit faults on average (for $b = 1$).

Table 2.8: Comparison between yield improvements achieved with BR and WR for yield=0.5 and b=1.

	# of Exchanged CCSBs					
	1	2	3	4	5	6
BR	2,302	3,577	4,799	6,126	7,257	8,398
WR	2,236	3,374	4,469	5,514	6,465	7,347
(BR – WR)	0,067	0,203	0,330	0,612	0,792	1,052

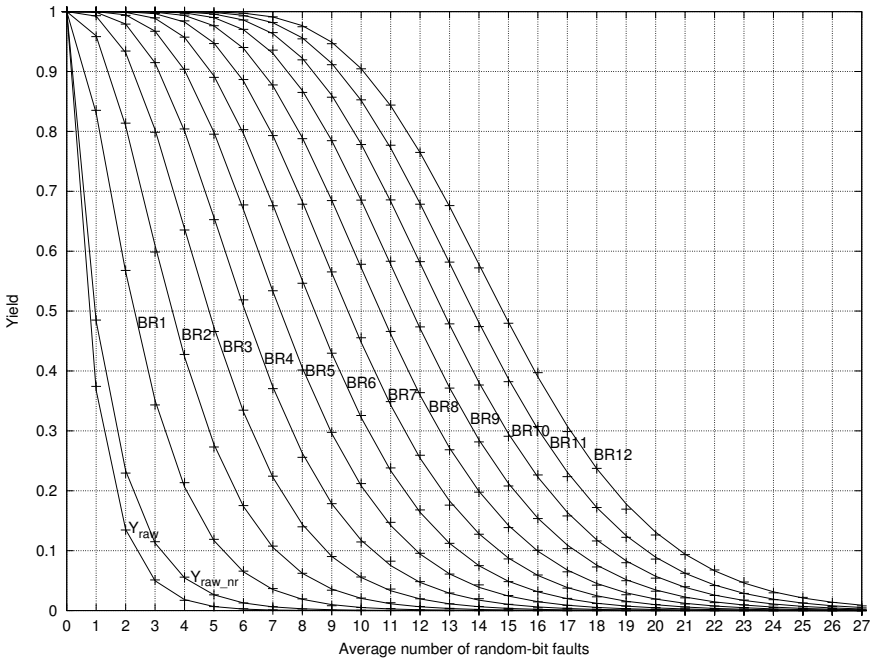


Figure 2.17: Yield of the memory array with different implementations of the BR (b=1).

In Figures 2.18, 2.19, and 2.20, the results concerning yield improvements achieved with $BR12$ implemented for different number of blocks in the memory are presented[10]. Of course, the bigger is the number of memory blocks the better is the resulting yield improvement.

As the number of memory blocks increases, the area required for storing the configuration data also increases. For a single block where BR12 is implemented 84 bits are required to store 12, 6-bit EPPs, and one 12-bit CSP vector. The total area required for the BR12 implemented for the example memory ($b = 1$) including the area required for

[10]Due to long simulation times, only results calculated with yield formulas are reported.

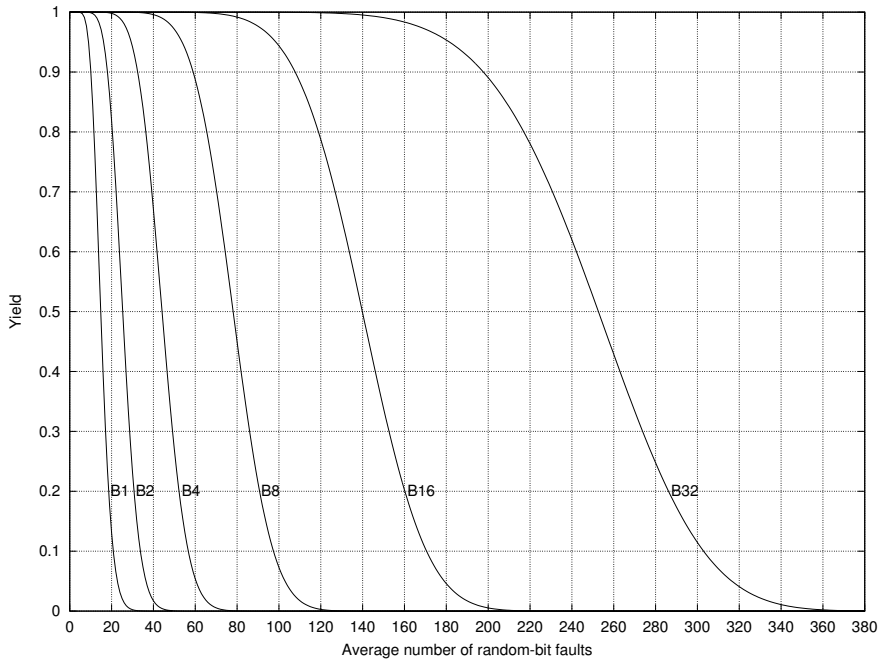


Figure 2.18: Memory array yield improvement achieved with the BR12 technique implemented for different number of memory blocks (1-32).

SPs is $84 + 8192\ words\ \times\ 12\ EPPs\ \times\ 1\ CSB\ for\ each\ SP\ = 98388$ bits which is $\approx 27,3\%$ of the example memory area. The area required for different number of memory blocks is presented in Table 2.9.

Table 2.9: Area required for the BR12 implemented for different number of memory blocks.

# of Blocks	Area Overhead (bits)		Ratio (%)	
	CFG Data	SPs	CFG Data to SPs	(CFG Data + SPs) to Memory Area
1	84	98304	0,085	27,296
2	168	98304	0,171	27,319
4	336	98304	0,342	27,366
8	672	98304	0,684	27,459
16	1344	98304	1,367	27,646
32	2688	98304	2,734	28,018
64	5376	98304	5,469	28,764
128	10752	98304	10,938	30,256
256	21504	98304	21,875	33,239
512	43008	98304	43,750	39,205
1024	86016	98304	87,500	51,136
2048	172032	98304	175,000	75,000
4096	344064	98304	350,000	122,727
8192	688128	98304	700,000	218,182

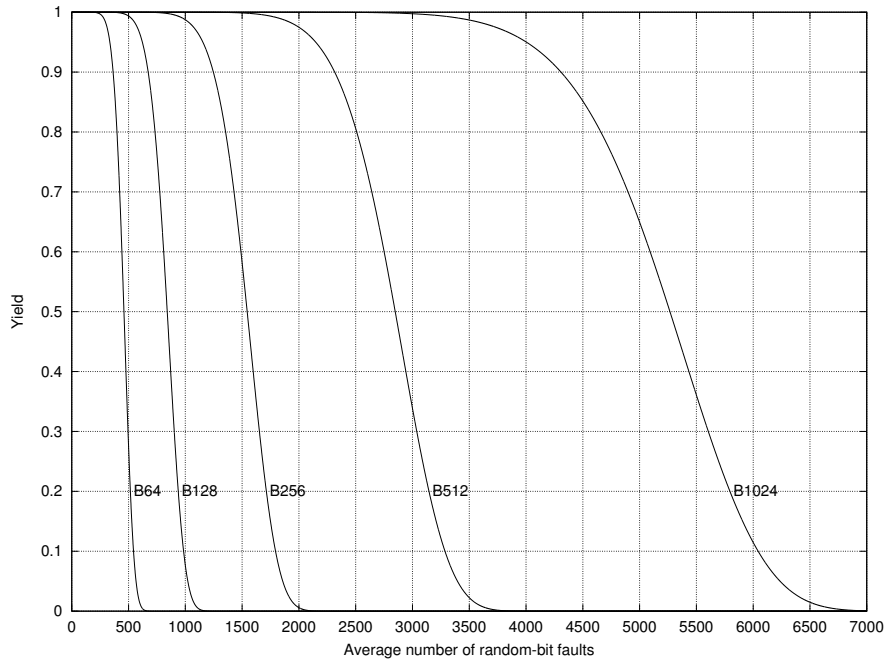


Figure 2.19: Memory array yield improvement achieved with the BR12 technique implemented for different number of memory blocks (64-1024).

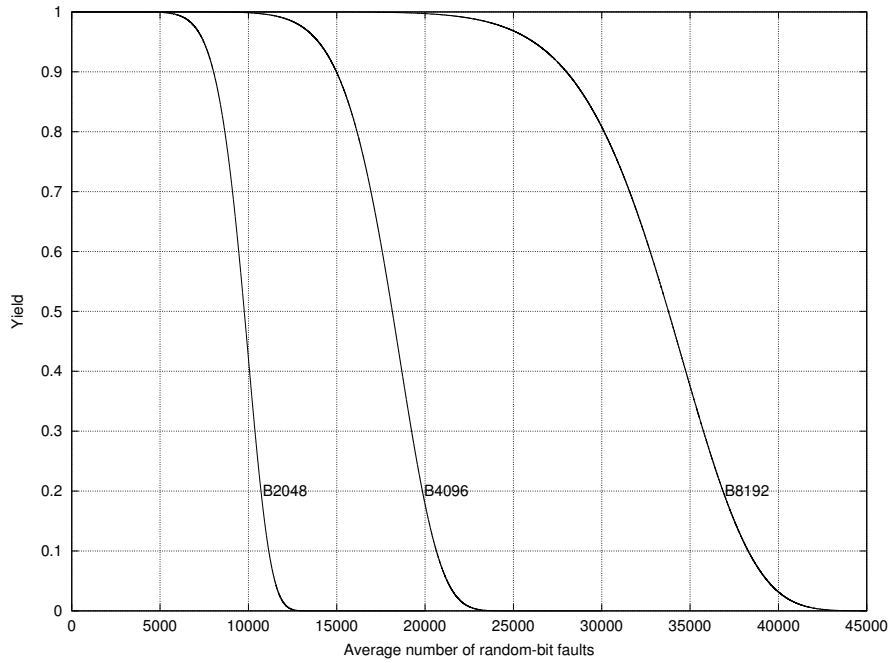


Figure 2.20: Memory array yield improvement achieved with the BR12 technique implemented for different number of memory blocks (2048-8192).

2.3.3 Word- and block-level repair

The word-level repair provides instant repair mechanism but requires two CSBs for every SP. On the other hand, the BR requires smaller spare area for replacing the same amount of CCSBs as the WR but suffers from limited on-line repair capability. Since WR and BR are based on similar principles, they can be combined into a consistent system. By doing so, an unprecedented repair mechanism can be achieved where the repair time, effectiveness, imposed meta-bit overhead and impact on the memory behavior can be tuned to the actual needs of target application.

There are many ways how the WR and BR techniques can cooperate. Depending on which repair technique (WR or BR) is selected for managing first faults in the memory array different repair schemes can be achieved. More information about possible combinations of the WR and BR techniques is presented in Section 4.2. In the thesis, combination of the WR and BR techniques where BR is used for first faults in the memory will be evaluated. The resulting repair system will be further referred as BWR.

The BWR system is aimed at memories which suffer from low yield, i.e., large number of post-production faults. The reason for this is that BR has limited on-line repair capabilities and it is preferred to use it mostly during post-production test and repair phases. Once the post-production faults are managed by the BR, WR technique, which excels in on-line repair, can be used during normal memory operation for managing endurance-related faults. As a consequence, the BWR repair system seems suitable for emerging NVMs since they suffer from immature fabrication processes.

The BWR repair mechanism able to replace $(n + m)$ CCSBs requires (Fig. 2.21):

- n BR EPPs and n BR SPs which belong to the BR,
- m WR EPPs and m WR SPs which belong to the WR, and
- $(n + m)$-bit CSP vector for marking corrupted SPs,

for each memory block.

The flow chart diagram presenting the BWR mechanism is depicted in Figure 2.22. The verification and evaluation phases are the same as in WR and BR. The difference lies in the repair phase. Once the CCSBs will be detected in the user area, the standard BR procedure is executed. The BR procedure manages CCSBs until all SPs which belong to the BR are used. Next, if there are no free BR SPs, the WR procedure is used for managing memory array faults. If all WR SPs are used, no new CCSBs can be replaced.

Because SPs belonging to the WR have higher indexes, they have precedence over BR SPs. As a result, the WR is able to replace CCSBs in the user area and in the SPs belonging to the BR.

In the example memory following configurations of the BWR can be implemented:

- BR10WR1, where 10 CCSBs can be replaced by the BR and 1 CCSB by the WR,
- BR8WR2, where 8 CCSBs can be replaced by the BR and 2 CCSBs by the WR,
- BR6WR3, where 6 CCSBs can be replaced by the BR and 3 CCSBs by the WR,

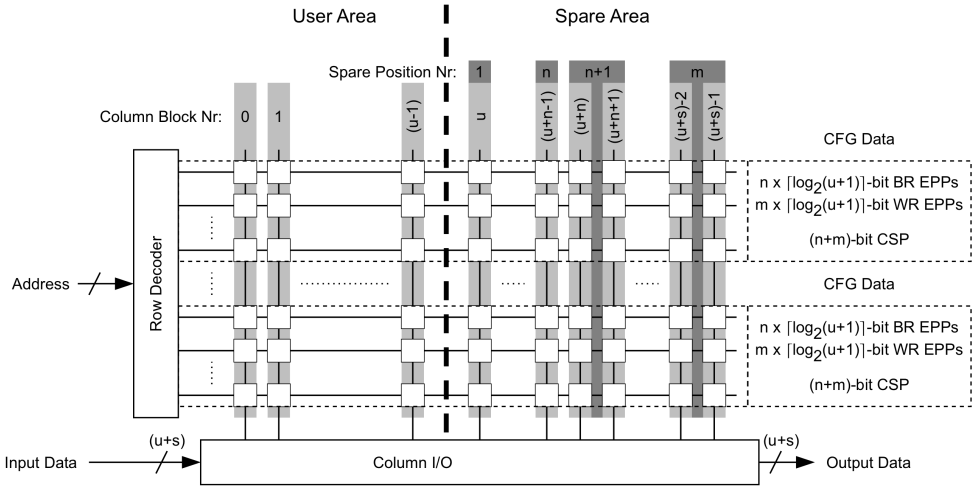


Figure 2.21: Layout of the memory array with required configuration data for the BWR.

- BR4WR4, where 4 CCSBs can be replaced by the BR and 4 CCSBs by the WR, and
- BR2WR5, where 2 CCSBs can be replaced by the BR and 5 CCSBs by the WR.

The yield improvement achieved with different configurations depends only on the number of CCSBs which can be replaced. As a consequence, the following formula can be used for calculating the yield improvement for a single memory block (where $x = (n+m)$ is the number of CCSBs which can be exchanged):

$$Y_b = \sum_{i=0}^{x} \binom{44}{i} \times Y_{csb}^{(44-i)} \times (1 - Y_{csb})^i \tag{2.15}$$

In Figure 2.23 results obtained from yield formulas and computer simulations are presented for different configurations of the BWR and $b = 1$. The simulations were performed in a similar approach as simulations used for the WR and BR evaluation.

The main conclusion which can be made based on the results provided in Figure 2.23 is that the lower is the number of CCSBs which can be replaced, the lower the memory yield improvement can be achieved.

As presented in the previous section, BR able to replace the same number of CCSBs as WR achieves better memory yield improvement. The same observation can be made in the BWR system. In Figure 2.24 comparison between configurations of the BWR and BR able to replace the same number of CCSBs is presented. The results for the Figure 2.24 where obtained using yield formulas.

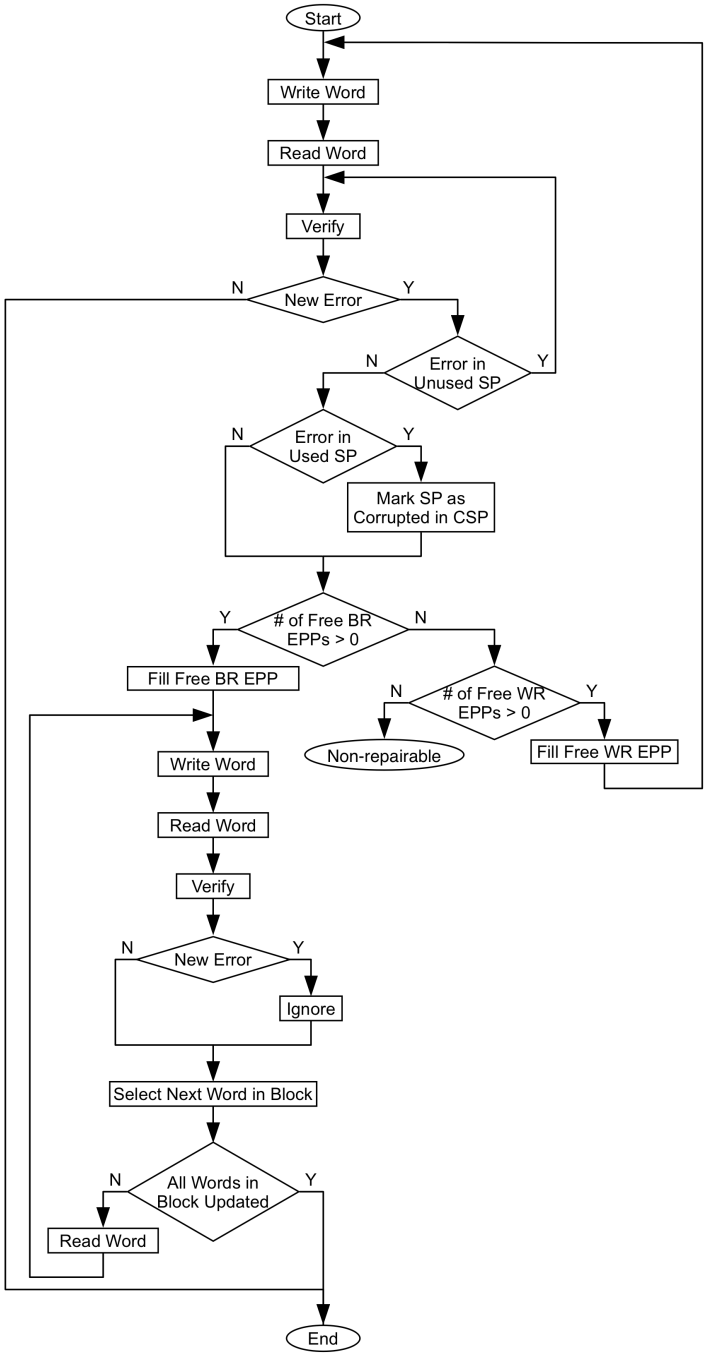


Figure 2.22: Flow chart of the BWR mechanism.

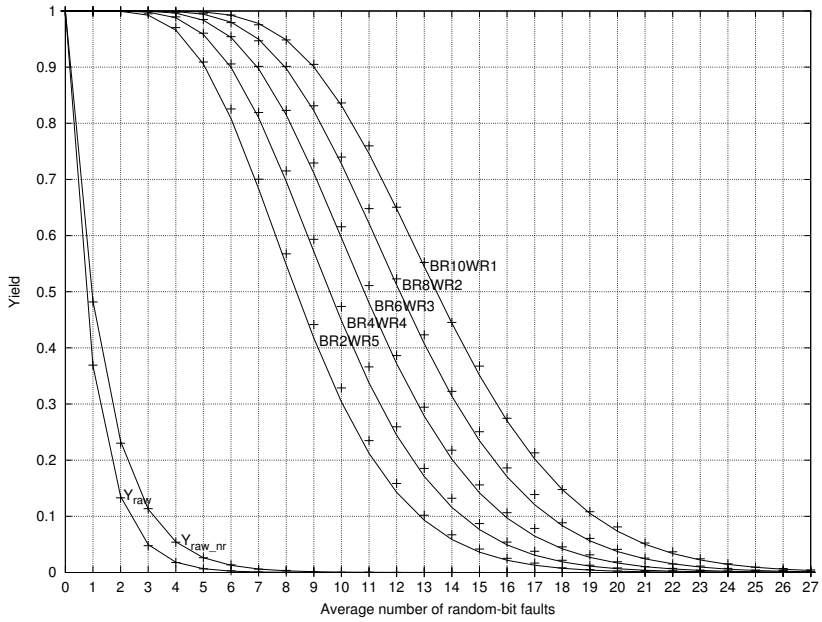


Figure 2.23: Yield of the memory array with different configurations of the BWR (b=1).

The BR implementations able to replace the same number of CCSBs as BWR achieve better memory yield improvement. The lower yield achieved by the BWR is caused by the WR implementations which require 2 CSBs for every SP. For example note results for the BR4WR4 and BR7. The BR7 configuration able to replace 7 CCSBs achieves similar memory yield improvement as BR4WR4 configuration able to replace 8 CCSBs.

The additional area required to implement the BWR depends on numbers of CCSBs which can be replaced by the BR and by the WR. For example, for the BR10WR1 configuration, 77 bits are required for 10 BR EPPs, 1 WR EPP, and 11-bit CSP vector. The total area required for the BR10WR1 implemented for the example memory ($b = 1$) including the area required for SPs is $77 + 8192\ words\ \times\ (10\ BREPPs\ \times\ 1\ CSB\ for\ each\ BR\ SP\ +1\ WREPP\ \times\ 2\ CSBs\ for\ each\ WR\ SP)\ = 98381$ bits which is $\approx 27,3\%$ of the example memory area. The area required for different BWR configurations implemented for the example memory ($b = 1$) is presented in Table 2.9.

Table 2.10: Area required for different configurations of the BWR implemented for the example memory (b=1).

BWR Configuration	Area Overhead (bits)		Ratio (%)	
	CFG Data	SPs	CFG Data to SPs	(CFG Data + SPs) to Memory area
BR10WR1	77	98304	0,078	27,294
BR8WR2	70	98304	0,071	27,292
BR6WR3	63	98304	0,064	27,290
BR4WR4	56	98304	0,057	27,288
BR2WR5	49	98304	0,050	27,286

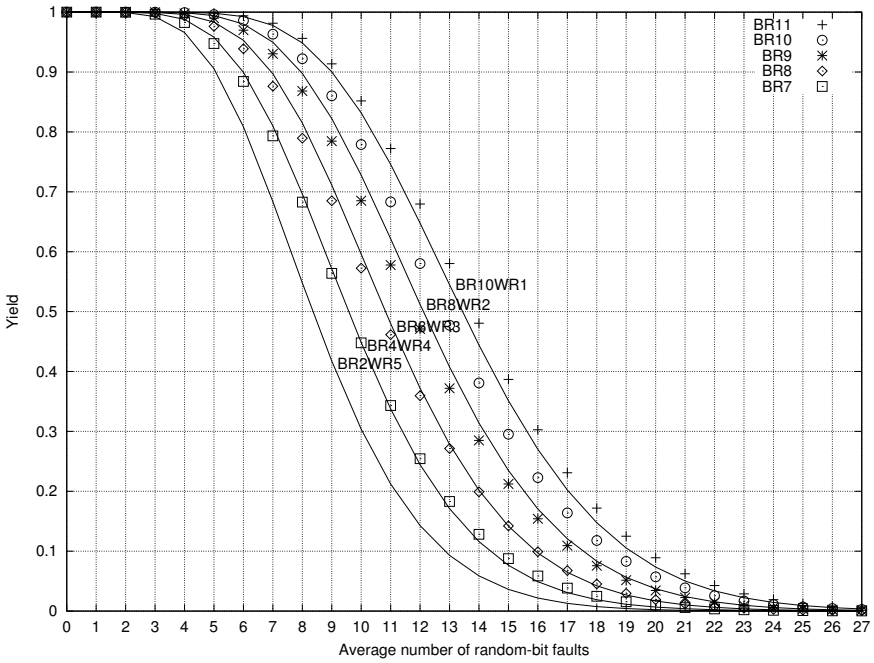


Figure 2.24: Comparison between different configurations of BWR and BR.

Comparison with other redundancy-repair techniques

The repair system based on the combination of the WR and BR was compared with other on-line redundancy repair techniques in [90]. Further in the thesis results from mentioned evaluation will be presented.

In the evaluation process, the WBR[11] was compared with SAFER [84], ECP [81], and ECC technique. Moreover, to have most comparable results a similar evaluation methodology to the methodologies used in [84] and [81] was exploited. Therefore, in the evaluation process the following assumptions were made:

- The repair procedures are assumed to be implemented in the non-volatile memory which does not require erase-before-write operation.
- The lifetime of each memory cell follows the Normal distribution with a mean lifetime of 10^8 and standard deviation equal to 10^7. Moreover, no correlation between neighboring sells is assumed.
- For every simulation it is assumed that perfect wear-leveling mechanism is implemented which evenly distributes the wear among all blocks in the memory. Moreover, similarly as in [84], all evaluated techniques are implemented for a single 2048-bit memory block.

[11] Repair system based on the BR and WR where first faults in the memory array are repaired by the WR. When all WR SPs are used, the following memory array faults are repaired by the BR.

- The toggle rate for each data memory cell is assumed to be 0.5. The toggle rate for meta-bits will be further described for each technique separately.
- The maximum meta-bit overhead for each technique is assumed to be less than 12,5% of the memory block size.

In the experiments, each technique was implemented for different word sizes which varied from 16 to 512 bits. Each test configuration consisted of one repair technique able to correct maximum number of faults for a selected word size imposing meta-bit overhead less than 12,5%. In a single simulation, array required for storing data bits and meta bits was allocated. For each array element random write endurance value was set using the Normal distribution and appropriate toggle rate value was assigned. Each simulation loop consisted of:

- selecting a failed cell from the allocated array i.e. i^{th} array element with the minimum write endurance in respect to the element's toggle rate value:

$$min\_end_i = \frac{i^{th} write\ endurance}{i^{th} toggle\ rate} \tag{2.16}$$

- decreasing the write endurance value of each array element by min_end of the failed cell,
- performing repair operation, and
- assigning and updating toggle rate values depending on the used technique.

Simulation was ended when the repair procedure could not be further performed. For each configuration 50000 simulations were performed and average result was reported.

IdealECC - The ECC technique is based on adding redundancy (in form of parity bits) to the memory input data in order to restore the original information in case of errors. For the purpose of the evaluation process, Hamming bound was used to determine the minimum number of parity bits required to correct x errors. The main reason of using the Hamming bound is that it defines a theoretical limit of techniques based on ECCs. As a consequence, the IdealECC refers to the theoretical ECC-based repair. Known implementations of ECCs may require larger number of parity bits than the number of parity bits calculated with the Hamming bound.

Since the error repair procedure is based on previously calculated parity bits, before each simulation, the toggle rate assigned for meta-bits is the same as for data bits. The technique based on ideal coding will be further referred as $IdealECCx$ where x defines the number of correctable errors per memory word.

ECP - The ECP presented in [81] and described in Section 1.2.2 is based on exchanging failed memory cells with additional replacement cells. In order to provide the repair procedure, the ECP is using addresses of defective cells determined during verification

process (write-verify scheme). The ECP can correct multiple hard errors including errors in its own structures. Technique based on the ECP will be further referred as $ECPx$ where x defines the number of errors which can be corrected in the memory word.

In the ECP, error pointers change only when new errors occur. As a consequence, replacement values, once they are assigned, change with the same frequency as data bits. Therefore, in the simulation, the toggle rate for replacement memory cells is set to 0.5 once the replacement cells are assigned.

SAFER - The SAFER is based on the observation that cells with stuck-at faults are readable and can be used for storing values. The repair procedure presented in [84] and described in Section 1.2.2 is performed by dynamically repartitioning memory word into separate groups where each group has at most one failed cell. Next, depending on the stuck-at value of failed cell located in the group, the input data for that group is stored in the original or inverted form. Repartitioning is performed every time new error occurs. In the SAFER technique information about error positions or their stuck-at values is not stored. Therefore, when data is stored to the corrupted memory word, additional write is needed every time the input data does not match the stuck-at values.

Further, $SAFERg$ defines the SAFER technique capable of repartitioning memory word into g groups. In the SAFER technique $\log_2 g$ defines the maximum number of possible repartitions which can be performed on a single word. If further repartitions cannot be performed, new errors can be corrected only if they will occur in uncorrupted groups. Therefore, $SAFERg$ can correct from $\log_2 g + 1$ to g errors per memory word.

WBR - In the WBR system, BR procedure is performed after all WR SPs are used. Moreover, since SPs provide replacement bits, once they are assigned for replacement, they suffer from the same wear as data bits. Therefore, in simulations, when an appropriate SP is used for replacement, toggle rate equal to 0.5 is assigned to its bits.

According to the evaluation assumptions, the repair techniques are implemented for non-volatile memory which does not require erase-before-write operation. As a result, the BR implemented in the WBR does not require the block-erase procedure (see Section 2.3.2).

The meta-bit overhead imposed by the WBR strongly depends on the size of the memory block. Therefore, in the evaluation phase, the original 2048-bit memory block was divided into many virtual blocks for which the WBR was implemented.

The best results from different configurations of virtual block sizes are further reported. Moreover, because BR approach provides better memory lifetime improvement than the WR for comparable meta-bit overheads, configurations with second best results are also included. If multiple configurations shared similar results, the configuration with higher number of CCSBs which could be replaced by the WR was selected.

The $WBR\_b\_wr\_br$ defines a WBR approach where b is the virtual block size, wr is the number of CCSBs which can be replaced by the WR, and br is the number of CCSBs which can be replaced by the BR.

Evaluation results

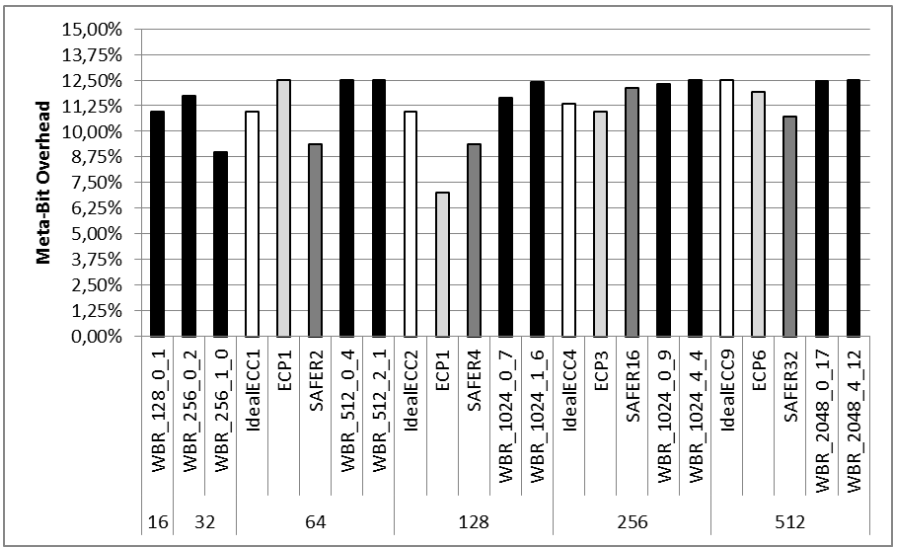


Figure 2.25: Meta-bit overhead imposed by different configurations of repair techniques.

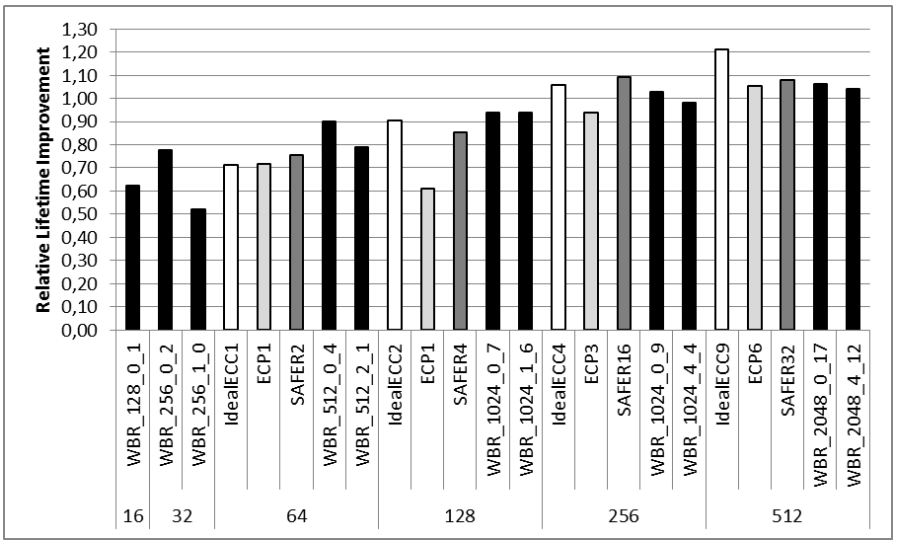


Figure 2.26: Relative lifetime improvement achieved with different configurations of repair techniques.

Meta-bit overhead - In Figure 2.25 and in Table 2.11 meta-bit overhead is presented for each repair technique. As one can observe, the WBR incurs meta-bit overhead close to the imposed restriction. The reason for this is that the WBR is very flexible when it comes to the requirement for additional area used for meta-bits. The additional area

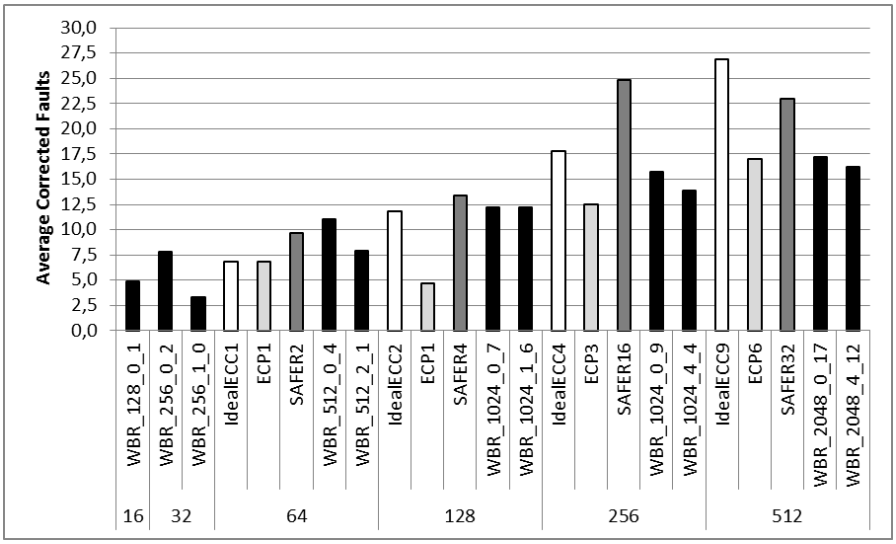


Figure 2.27: Average number of corrected faults for different configurations of repair techniques.

required for the WBR strongly depends on the configuration of WR and BR mechanisms. For example, for 512-bit words in a 2048-bit memory block, the IdealECC offers 9 repair configurations, ECP 6 configurations, SAFER 5 configurations, and the WBR almost 130 different repair configurations. Moreover, in contrast to other solutions, the WBR can provide repair mechanism even for 16-bit and 32-bit words imposing less than 12.5% of additional area.

Lifetime improvement - The main point of the evaluation is the memory lifetime improvement which can be achieved with different repair mechanisms. Similarly like in [84] the lifetime improvement is presented as a function of a standard deviation. The relative lifetime improvement used in the evaluation process is calculated as:

$$Relative\_improvement = \frac{(L - F) * T}{\sigma} \tag{2.17}$$

where L is the average number of writes to the memory block achieved with the help of repair mechanism, F is the average number of writes until the first fail in the memory block occurs, T is a toggle rate value, and σ is the standard deviation used in the evaluation. The relative lifetime improvement was calculated after each simulation. Next, after evaluation of a single configuration (50000 simulations) the average relative lifetime improvement was calculated and reported.

In Figure 2.26 and in Table 2.11 average relative improvement values are reported for different repair mechanisms. For word sizes up to 128 bits, the WBR outperforms not only SAFER and ECP-based solutions but also the IdealECC technique. For example, for 2048-bit memory block consisting of 128-bit words, the WBR increases memory life-

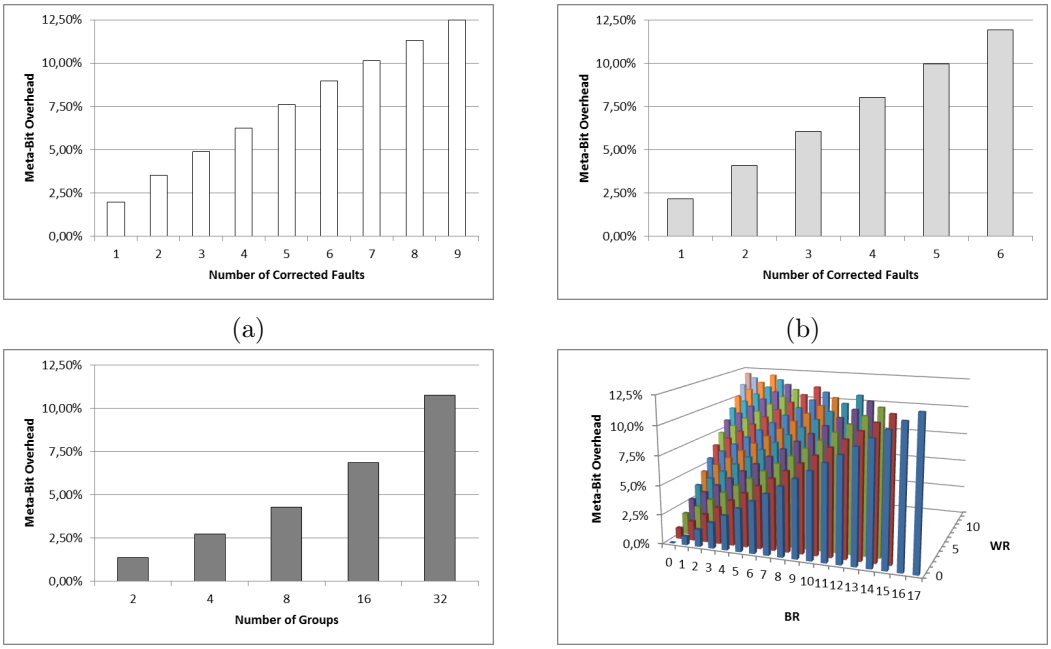


Figure 2.28: Meta-bit overhead of different repair configurations imposed by IdealECC (a), ECP (b), SAFER (c), and WBR (d).

time by $\frac{(0{,}9397*10^7)}{0{,}5} = 18794000$ writes, while SAFER increases memory lifetime by only $\frac{(0{,}8486*10^7)}{0{,}5} = 16972000$ writes. For memory word sizes greater than 128 bits, the WBR lies between ECP and SAFER.

Average number of corrected faults - Another interesting figure concerning repair mechanisms is the average number of corrected faults. From Figure 2.27 and Table 2.11 one can observe that the WBR achieves the highest average number of corrected faults for 16, 32, and 64-bit words while SAFER dominates in wider word sizes.

In general, results concerning average number of corrected faults achieved with WBR, ECC- and ECP-based techniques translate into average relative lifetime improvement values. The reason why the results for SAFER do not correspond to the average relative lifetime improvement is that, in SAFER, an additional write operation is required for each corrupted group after the repair procedure. These additional write operations decrease the lifetime of undamaged memory cells in the corrupted group and the effectiveness of the whole repair mechanism.

Table 2.11: Comparison results of different repair techniques.

Data Size	Recovery Scheme	Meta-bit Overhead	Avg. Corrected Faults	Avg. Relative Lifetime Improvement
16	WBR_128_0_1	10,94%	4,8738	0,6231
32	WBR_256_0_2	11,72%	7,8074	0,7783
	WBR_256_1_0	8,98%	3,3038	0,5206
64	IdealECC1	10,94%	6,8158	0,7126
	ECP1	12,50%	6,8169	0,7174
	SAFER2	9,38%	9,6931	0,7565
	WBR_512_0_4	12,50%	11,0786	0,9010
	WBR_512_2_1	12,50%	7,9042	0,7905
128	IdealECC2	10,94%	11,7927	0,9040
	ECP1	7,03%	4,7002	0,6117
	SAFER4	9,38%	13,3574	0,8516
	WBR_1024_0_7	11,62%	12,1702	0,9397
	WBR_1024_1_6	12,40%	12,1751	0,9381
256	IdealECC4	11,33%	17,7768	1,0557
	ECP3	10,94%	12,4857	0,9373
	SAFER16	12,11%	24,8401	1,0936
	WBR_1024_0_9	12,30%	15,7026	1,0267
	WBR_1024_4_4	12,50%	13,8370	0,9812
512	IdealECC9	12,50%	26,8828	1,2116
	ECP6	11,91%	17,0338	1,0545
	SAFER32	10,74%	22,9388	1,0781
	WBR_2048_0_17	12,45%	17,2289	1,0610
	WBR_2048_4_12	12,50%	16,2022	1,0414

2.4 Error-correcting code with increased hard-error correction capability

Error-correcting codes are widely utilized for improving the reliability of semiconductor memories. Depending on the application, memory technology and architecture, there are many ECCs able to provide required reliability level.

In conventional semiconductor memories ECCs are mostly used for managing radiation-induced soft errors while redundancy repair techniques are utilized for post-production hard faults occurring in the memory array. In emerging NVMs the need for protection against radiation-induced soft errors is not so great as for conventional memories. The main reason for that is their storage mechanism which is not based on charge storage as in SRAMs, DRAMs, or flash memories. Nevertheless, emerging NVMs suffer from retention- and endurance-related faults which can occur at any point during their lifetime. Moreover they are more vulnerable to environmental factors (e.g. ambient temperature, magnetic fields) than conventional memories. Therefore, for comprehensive reliability management aimed at non-volatile memories, ECCs are a very important component. However, instead of correcting soft-errors, ECCs implemented in the emerging NVMs should be focused on hard-error correction.

In general, ECCs can be defined by a $p \times n$ parity matrix H and three parameters (n, k, d) , where k is the number of data bits which should be protected, n is the number of encoded bits (data bits + parity bits), and d is a minimum Hamming distance of a code [16]. The number of required parity bits p is equal to $(n - k)$.

Usually, to facilitate the encoding process, parity matrix is expressed in a systematic form as $H = [H_0\ I_p]$, where H_0 is the $(p \times k)$ binary matrix used for generating parity bits and I_p is the $p \times p$ identity matrix [16]. In such case, the first k bits of a codeword are data input bits, and the last p bits are parity bits.

The first step in protecting data using the ECC is to generate parity bits during the encoding process [16]:

$$P_{in} = H_0 \cdot D'_{in} \tag{2.18}$$

Where D'_{in} represents transposed input data vector, P_{in} denotes calculated parity vector, and all additions are performed modulo 2. Next, parity bits together with input data bits are stored in the memory as a memory word $W_{in} = [D_{in}\ P'_{in}]$.

Due to memory faults or external factors data stored in the memory can be altered. Therefore, word read out from the memory $W_{out} = [D_{out}\ P'_{out}]$ can be different than the stored word W_{in}.

After reading a word from memory, first, parity bits are generated during decoding process [16]:

$$P = H_0 \cdot D'_{out} \tag{2.19}$$

Next, they are xor-ed with stored parity bits in order to obtain a syndrome [16]:

$$S = P \oplus P_{out} \tag{2.20}$$

If calculated syndrome S is an all0 vector, then the word read from the memory is assumed to be error free. If S is a non0 vector then the word is assumed to be faulty and S can be used to obtain the error vector [16].

There are few classes of codes well suited for memory systems such as Hamming for correcting single-bit errors or RS and BCH codes for multi-bit correction purposes [91]. In general, error correction capability of a code is defined by the minimum Hamming distance between any two codewords. Generally, t errors can be corrected and $(t + 1)$ errors can be detected, if and only if [16]:

$$d > 2 \times t + 1 \tag{2.21}$$

Unfortunately, stronger codes (such as RS and BCH) capable of correcting multi-bit errors require area, time, and power overheads which are hard or even impossible to accept in many embedded applications. Therefore, for latency constrained, random-access memories, single-bit error-correcting double-bit error-detecting (SEC-DED) codes[12] such as extended Hamming[13] codes are preferred. Because emerging NVMs are expected to

[12]SEC-DED codes have minimum Hamming distance of 4.

[13]Standard Hamming codes are SEC codes whose minimum Hamming distance is 3. Extended Hamming codes are capable of SEC-DED and their minimum Hamming distance is 4.

serve as RAMs, simple, low-latency ECCs are required. As a result, for the comprehensive approach presented in the thesis a Hsiao code[14] will be used [41]. For the example memory (39, 32, 4) configuration of the Hsiao code will be implemented. In order to construct such a code [41]:

- All columns in the parity matrix must be non0 columns.
- Every column has to be distinct.
- Every column must contain an odd number of 1's (hence the name "odd-weight").

As described in [41], the first two constrains give a distance 3 code. The third constraint guarantees distance 4 code, because the modulo 2 sum of any three odd-weight columns never equals 0. In principle, the modulo 2 addition of any odd number of odd-weight vectors will always give an odd-weight vector while any even number of odd weight vectors will give an even-weight vector including the all0 vector.

The biggest advantage of the Hsiao code in comparison to the extended Hamming code is that the number of 1's in the parity matrix is reduced. Smaller number of 1's relates to a smaller number of logic levels required to perform the calculation of parity bits. In addition, less hardware implies better reliability and imposes shorter delay [41]. In order to construct a parity matrix with optimal number of 1's for predefined number of data bits [41]:

- Every column should have an odd number of 1's; i.e., all column vectors are of odd weight.
- The total number of 1's in the H-matrix should be a minimum.
- The number of 1's in each row of the H-matrix should be made equal, or as close as possible to the average number i.e., the total number of 1's in H divided by the number of rows.

The parity matrix of the (39, 32, 4) configuration of the Hsaio code is presented in 2.22.

$$H = \begin{bmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&1&0&0&0&0\\
1&0&0&0&1&0&0&0&1&1&1&0&0&0&1&0&0&0&1&1&1&0&0&0&1&1&1&0&0&0&1&1&0&0&0&1&0&0&0\\
0&1&0&0&0&1&0&0&1&0&0&1&1&0&0&1&0&0&1&0&0&1&1&0&1&0&0&1&1&0&1&1&0&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&1&0&1&0&1&0&0&1&0&0&1&0&1&0&1&0&1&0&1&0&1&1&0&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&1&0&0&1&0&1&1&0&0&0&1&0&0&1&0&1&1&0&0&1&0&1&1&0&1&0&0&0&0&0&0&1
\end{bmatrix} \tag{2.22}$$

Since the Hsiao code is an error correcting block code, decoding process is the same as previously presented. If the calculated syndrome S (Eq. 2.20) is an all0 vector then

[14]Hsiao code is a special version of the extended Hamming code. It is an odd-weight, SEC-DED code.

the word read from the memory is assumed to be error free. In other case, if the number of 1's in the syndrome S is odd there is an assumption that a single-bit error occurred (although the word read from the memory may contain an odd number of errors). In this case, an appropriate column in the parity matrix H which is equal to S refers to the position of the corrupted bit. If the number of 1's in the syndrome is even, then there is an assumption that double-bit error occurred (although the word read from the memory may contain an even number of errors) [41].

The ability of correcting random-bit errors is more expensive in terms of required number of parity bits than detecting errors. Contrary to the correction mechanism, error detection procedure does not require positions of errors. If there is an information about positions of potentially erroneous bits, correction capabilities of ECCs can be extended.

The idea of potentially-erroneous bits with known locations is well-known in the coding theory and those potentially-erroneous bits are called *erasures* [97]. This concept of erasures is employed in digital communications and large-capacity distributed storage systems, but surprisingly is not so commonly used in semiconductor memory systems [3].

There are many types of faults in existing and emerging memories which affect certain parts of the memory array (e.g. coupling or stuck-at faults). In principle, memories with those faults can be modeled as memories with areas where the probability of an error is higher than in the rest of the memory. This is a kind of situation where the concept of erasures can be used.

From the coding theory it is known that a code with minimum Hamming distance d can correct t random errors and r erasures if [97]:

$$d > 2 \times t + r \tag{2.23}$$

Therefore, if we know that errors can occur only in certain positions in the memory we can introduce $t = 0$ to the above equation. If all memory locations have equal probability of an error then $r = 0$. In case of mixed situations we can choose appropriate values of t and r.

In [98] Walker, Sundberg and Black presented a decoding algorithm together with decoder architecture capable of correcting one erasure and one random, single-bit error. In their approach called *Single Error and Single Erasure Correction* (SEEC), when a single-bit error occurs first time in a memory block, the syndrome corresponding to that error (S_{stored}) is stored (Fig. 2.29a). Next, when a double-bit error is detected in the same memory block, previously stored syndrome (S_{stored}) is used for automatic correction of double-bit error assuming that the second error is caused by the erasure. The double-bit error correction can be described as follows [98]:

- A double-bit error is detected in a word read from the memory.
- The calculated syndrome S (Eq. 2.20) corresponds to a double-bit error. Calculate $S_{single} = S \oplus S_{stored}$ which corresponds to a random, single-bit error.
- Search in the parity matrix H for columns equal to S_{stored} and S_{single}. Positions of found columns define positions of errors in the output word. Correct errors by

flipping appropriate bits in the output word.

As stated in [98] "with the SEEC, virtually all double errors in a semiconductor memory can be automatically detected and corrected". Of course this is true only if syndrome S_{stored} is stored. If the double-bit error occurs and there is no previously stored syndrome, correction cannot be performed.

Previously described method can be improved if the position of an erasure, instead of its syndrome, is stored. This improvement was proposed by the author of this thesis in [89]. For (39, 32, 4) Hsiao code the syndrome of an error consists of 7 bits, whereas only $\lceil log_2(39+1) \rceil = 6$ bits are needed to store the position of an erasure in a 39-bit codeword. The position is stored in the same way as position of CCSB in an EPP. That is, the bit positions in a codeword are indexed starting from 1. The position of an erasure is stored in the ERA_POS vector. If the ERA_POS vector is an all0 vector then the ERA_POS is assumed as empty.

This little improvement can reduce storage overhead, especially when there is a large number of memory blocks [89]. Moreover, the storage required for the ERA_POS can be even further reduced if only certain parts of the memory area are expected to have higher probability of hard faults. For example, let us assume that due to immature memory fabrication process and/or circuit design errors post-production hard faults can occur only in the first half of the memory matrix. As a consequence the ERA_POS vector size could be reduced to consider only erasures occurring in the first half of the memory matrix.

Table 2.12 presents number of bits required to store a syndrome of an erasure and its position for different memory word sizes.

Table 2.12: Number of bits required to store the syndrome of an erasure and its position for different word sizes.

Word Size	(39, 32, 4) Hsiao Syndrome	ERA_POS
8	5	4
16	6	5
32	7	6
64	8	7
128	9	8
256	10	9
512	11	10

Further, the improved version of the SEEC will be referred as ECCI. The correction steps of ECCI are similar to the SEEC and are performed as follows (Fig. 2.29b):

- A double error is detected in a word read from the memory.
- The syndrome S corresponding to the received word corresponds to a double error. Find $S_{erasure}$ which is equal to the column in the H indicated by the erasure position. Calculate $S_{single} = S \oplus S_{erasure}$ which corresponds to the single random error.

- Search in the parity matrix H for columns equal to $S_{erasure}$ and S_{single}. Positions of found columns define positions of errors in the output word. Correct errors by flipping appropriate bits in the output word.

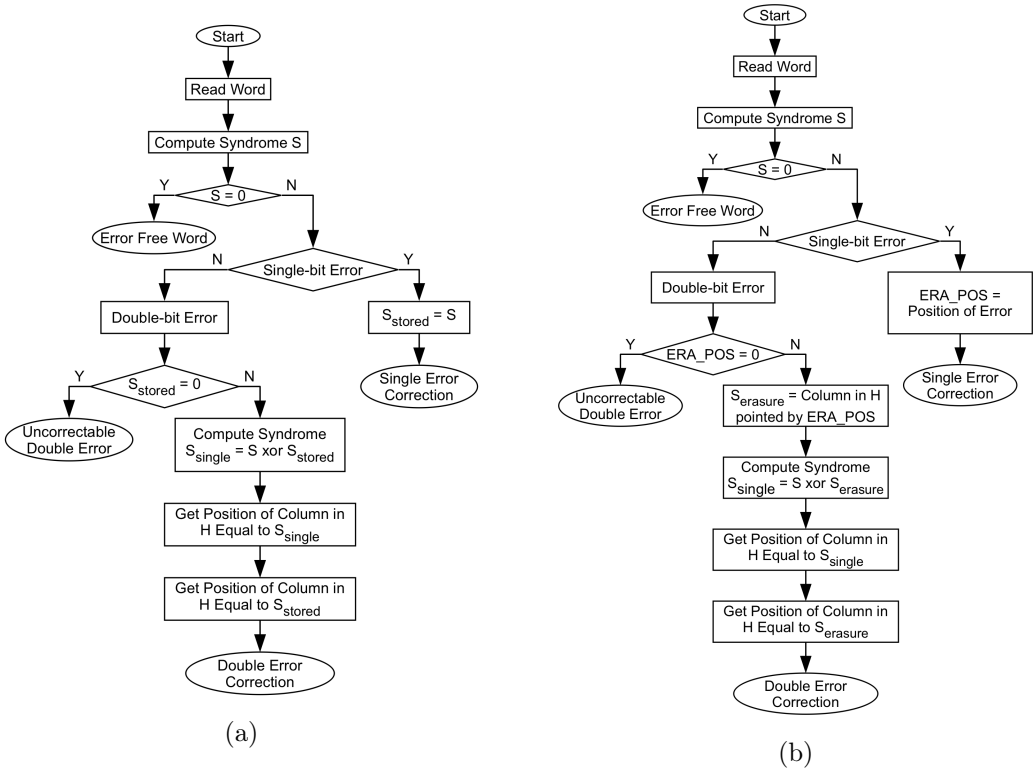


Figure 2.29: Flow charts presenting SEEC (a) and ECCI (b) approaches.

The yield improvement achieved with SEEC and ECCI is the same because they are based on the same repair mechanism. The difference lies only in the way how the information about the erasure is stored.

The results concerning the yield improvement achieved with ECCI implemented for the example memory and different block sizes are presented in Figures 2.30 and 2.31. Figure 2.30 presents results from computer simulations only. Because the readability of the Figure 2.30 is poor, the results obtained from computer simulations were approximated and depicted in Figure 2.31.

As one can observe, the yield improvement achieved with ECCI is higher in comparison to the yield improvement achieved with a standard implementation of (39, 32, 4) Hsiao code (referred as ECC in Fig. 2.31). Moreover, the yield improvement achieved with ECCI increases as the memory array is divided into smaller memory blocks. Since the position of an erasure is stored for each memory block independently, the bigger is the number of memory blocks, the larger number of erasures can be handled.

The repair mechanism of the ECCI is similar to the repair mechanism based on the

combination of the standard (39, 32, 4) Hsiao code and BR able to replace a single CCSB in a memory block (BR1). That is, in the ECCI, each memory word is protected against random, single-bit error. On top of that, a single erasure in a memory block can also be handled. As a consequence, once the position of an erasure is stored, hard faults occurring at the same position as the position of the erasure in other words in the memory block are also handled.

In the combination of the Hsiao code and BR1 each memory word is also protected against random, singe-bit errors. Moreover, a single CCSB can be replaced in the memory block. That is, faults occurring at the same position as the position of a CCSB are also handled.

Surprisingly, in the ECCI a synergistic effect can be observed. That is, the yield improvement achieved with ECCI is higher than the sum of yield improvements achieved with standalone implementations of the Hsiao code and the BR1 (Fig. 2.32).

Table 2.13 shows the average number of faults which can be corrected with standalone implementations of Hsiao code (ECC), BR1, and ECCI in the example memory ($b = 1$ and $yield = 0.5$). The column labeled *ECCI-(ECC+BR1)* shows the difference between the average number of faults corrected by the ECCI and the combined number of faults corrected by standalone implementations of the Hsiao code and the BR1.

As one can observe in Table 2.13 and Figure 2.33, if the memory array is divided into larger number of memory blocks then the resulting synergistic effect is better. The reason why the number of average corrected faults for $b = 2$ and $b = 3$ is lower than for $b = 1$ is not yet known and requires further research.

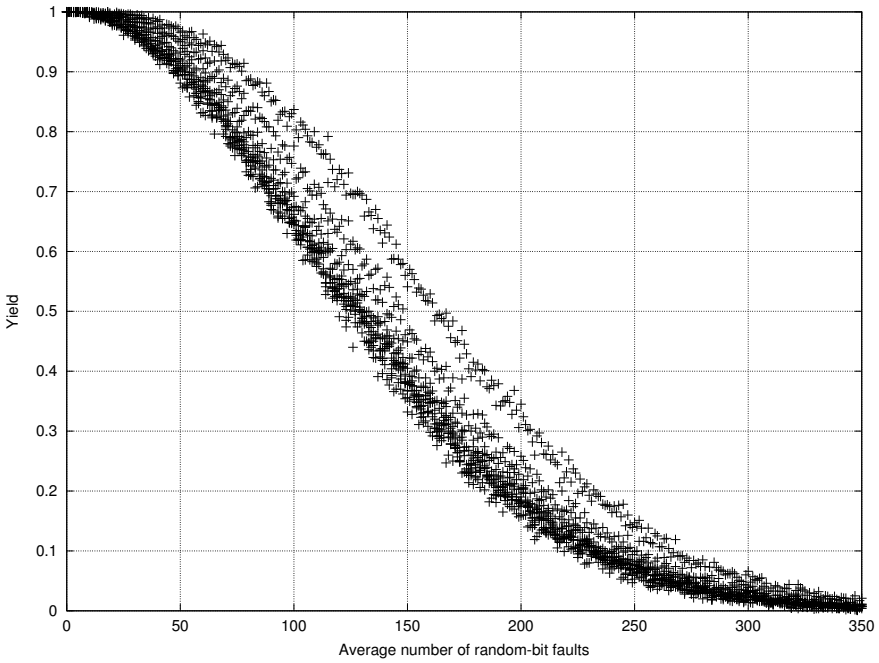


Figure 2.30: Simulation results concerning yield improvement achieved with the ECCI.

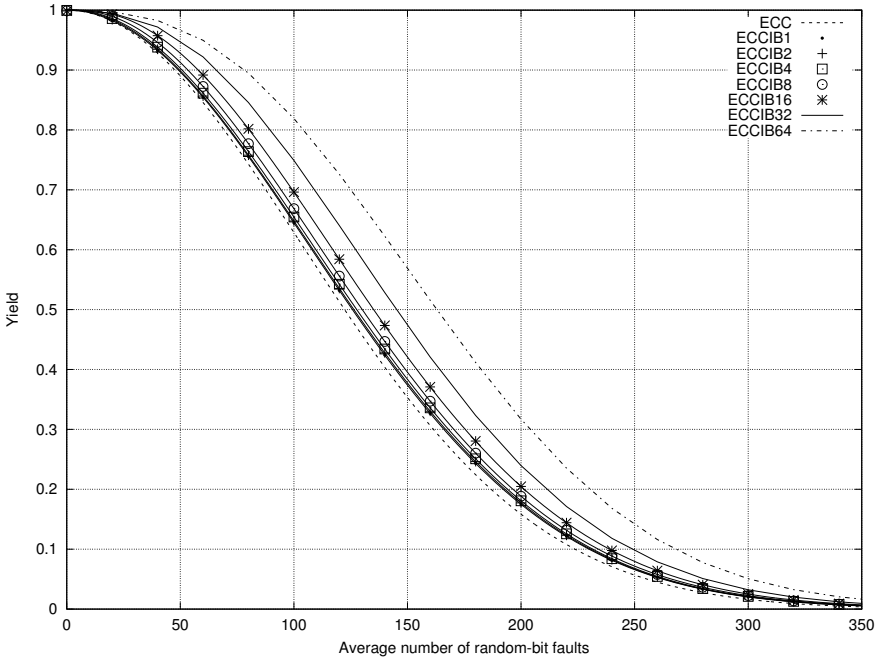


Figure 2.31: Approximated results concerning yield improvement achieved with the ECCI.

Table 2.13: Synergy effect achieved with ECCI.

# of Blocks	Average Number of Random-bit Faults (Yield = 0.5)			
	ECC	BR1	ECCI	ECCI-(ECC+BR1)
1	121,588	1,884	128,667	5,194
2	121,588	2,435	127,000	2,977
4	121,588	3,177	125,182	0,417
8	121,588	4,255	131,000	5,157
16	121,588	5,780	133,000	5,631
32	121,588	7,945	146,571	17,039
64	121,588	11,015	165,941	33,338

The synergistic effect of the ECCI is limited by the so-called *birthday paradox* [94]. The paradox gets its name from the fact that among 22 randomly selected people there is a better than 50% chance that two of these people have their birthday on the same day. For 40 people this chance increases to 90%. For 70 people it is almost certain that such pair of people will be found.

In semiconductor memory where ECC is implemented, the number of random-bit faults corresponds to the number of people in the random sample for the birthday paradox [94]. The number of codewords in the memory corresponds to the number of birthdays in a year. The occurrence of two or more faults in a codeword is comparable to the coincidence of two or more people having their birthday on the same date.

Since the ECC is able to correct only one error in a memory word, the occurrence of two

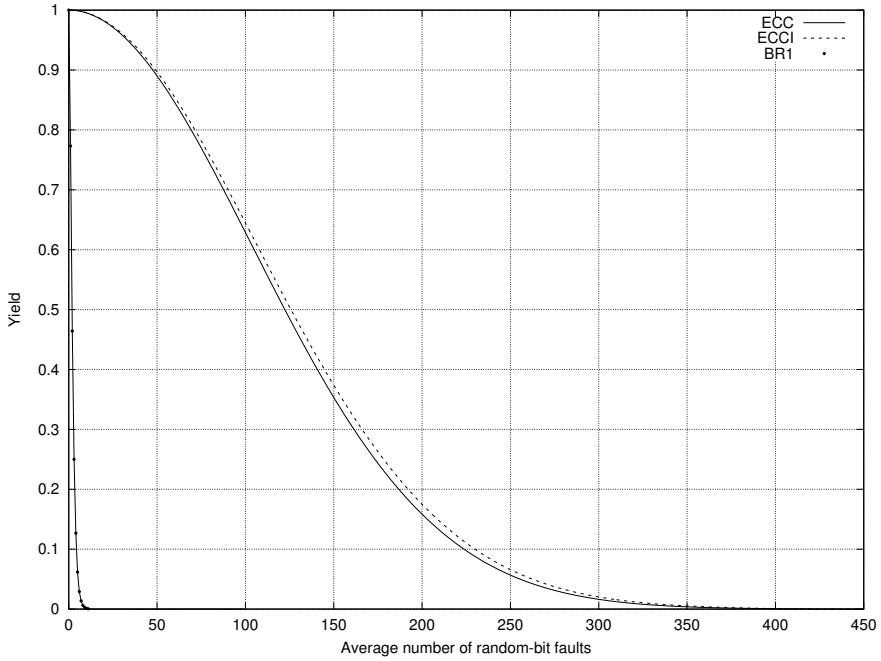


Figure 2.32: Memory yield improvement achieved with BR1, ECC, and ECCI.

or more faults in a codeword is unrepairable. As a consequence, the repair effectiveness of the ECC-based repair is limited by the birthday paradox.

In the example memory there are 8192 memory words. If the memory is fault-free, the probability that the first failing cell will not occur in already corrupted word is equal to 1. For a second failing cell the probability that it will not occur in the same word as the first fault is 8191/8192. For a third failing cell the probability that it will not occur in already corrupted words is 8190/8192. The multiplication of these probabilities result in a probability that the first, second, and third faults will not occur in the same memory word. For N failing cells, the probability that a memory word will not have more than one fault is (probability of no alignment) [94]:

$$P_{na} = \prod_{i=1}^{N} \frac{8193 - i}{8192} \tag{2.24}$$

In the example memory where the ECC is implemented, for 107 failing cells there is a better than 50% chance that two such failing cells will not occur in the same memory word. The yield of the memory associated with all these defects would therefore be more than 50%.

The synergistic effect of the ECCI can be increased if the second failing cell could also be corrected. That is, the position of an erasure should be stored only when a second failing bit in a memory word is detected. The ECCI design where the position of an erasure is stored only when second failing cell is detected in the memory word will be

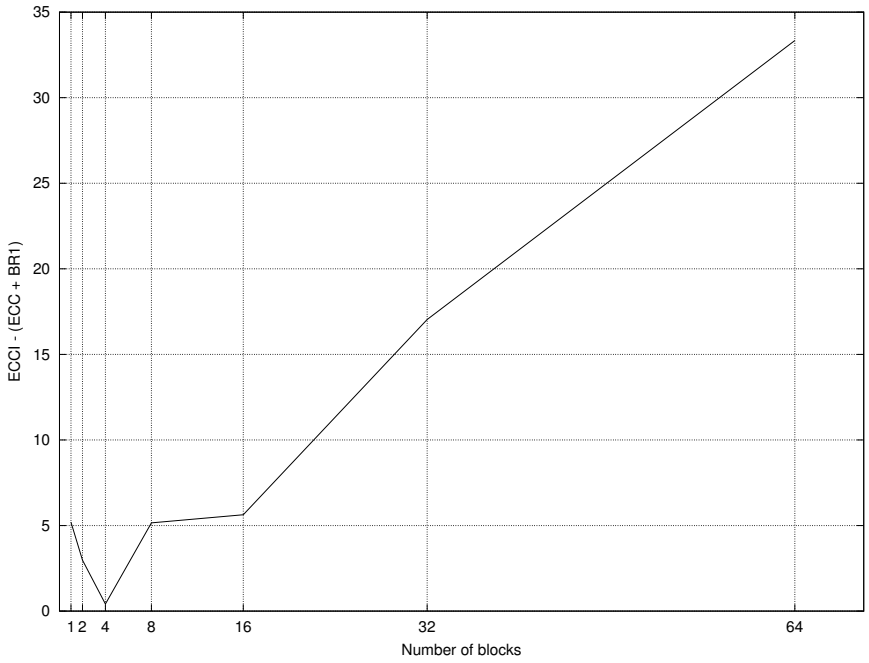


Figure 2.33: Synergistic effect achieved with the ECCI.

further referred as ERA.

In case of a double-bit error occurring in the memory word the implemented Hsiao code is unable to correct it. Moreover, the Hsiao code is able to detect double-bit error only, it cannot provide the error positions. As a consequence, another mechanism is required to provide the positions of faults in the memory word. Moreover, the position of the second failing cell in a corrupted word should be stored in ERA_POS before the corrupted word is read[15]. If we further assume that the ERA is aimed mostly at hard-faults, then the best candidate for providing fault positions in a memory word is already presented write-verify scheme. That is, whenever the memory word is written, it is read back and compared with the input data (Fig. 2.34b). Next, the difference vector is compared against the position stored in the enabled ERA_POS to distinguish new errors. Further, the number of new errors is counted. If the number of new errors is equal to 1 then no additional action is performed because the Hsiao code will correct it. If the number of new errors is equal to 2, then, if the ERA_POS is free, the position of one error is stored in the ERA_POS. The second error in a word is assumed to be corrected by the Hsiao code. If the ERA_POS is already used then faults that occurred in the memory word cannot be corrected and the resulting memory is defective. The same situation applies if the number of failing cells in the memory word is greater than 2.

The calculations of the yield improvement which can be achieved with ERA are not so easy to derive [40], [94]. The reason for this is that different scenarios of fault occurrences

[15]Because, as discussed, the Hsiao code is unable to correct double-bit errors.

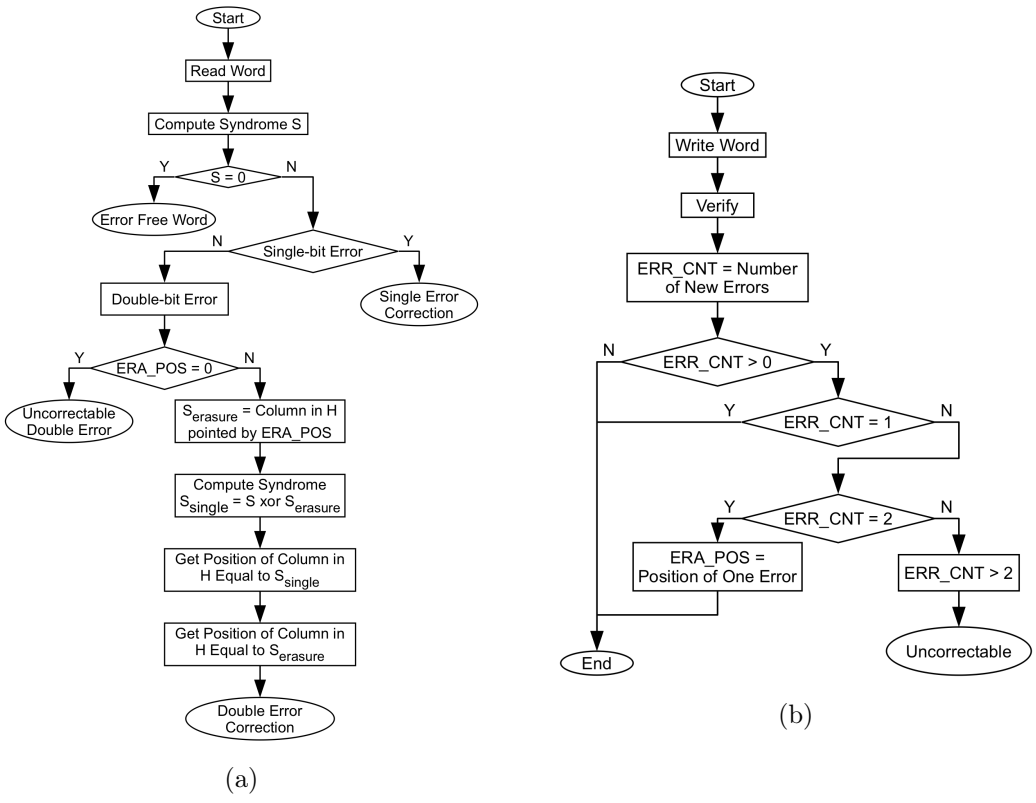


Figure 2.34: Flow charts presenting memory read (a) and write operations (b) managed by the ERA.

have to be considered. That is, in the example memory where the ERA is implemented [40], [94]:

- A codeword can have at most one faulty cell which will be corrected by the Hsiao code. The probability of such case is expressed by:

$$Y_{eccw} = Y_c^{39} + \binom{39}{1} \times Y_c^{38} \times (1 - Y_c) \tag{2.25}$$

- One codeword can have two failed cells. The position of one fault will be stored in the ERA_POS. Further, when the word will be read, two faulty cells will be corrected by the Hsiao code using ERA_POS (Fig. 2.34a). The probability that a codeword will have two faults can be calculated with:

$$Y_{df} = \binom{39}{2} \times Y_c^{37} \times (1 - Y_c)^2 \tag{2.26}$$

- If one codeword has two failed cells then the rest of codewords in the memory block can have no failing cells. In addition, they can have no more than one fault which

will be corrected with the Hsiao code, or they can have two failing cells. One failing cell can be corrected with the Hsiao code and a second failing cell can be located on the same position as already detected erasure. The probability of such case is:

$$Y_{cw} = Y_c^{39} + 39 \times Y_c^{38} \times (1 - Y_c) + 37 \times Y_c^{37} \times (1 - Y_c)^2 \tag{2.27}$$

As a result, the yield of memory block where ERA is implemented can be calculated with [40], [94]:

$$Y_b = Y_{eccw}^{W} + W \times Y_{df} \times Y_{cw}^{(W-1)} \tag{2.28}$$

Where $W = (8192/b)$ is the number of codewords in the memory block and $Y_c = \exp\left(-\frac{\lambda}{8192\times44}\right)$ is the yield of a single memory cell.

Figures 2.35 and 2.36 show the yield of the example memory where ERA was implemented for different number of blocks. The Figure 2.35 presents results obtained using the formula 2.28 while Figure 2.36 presents results obtained from computer simulations.

As one can observe in Figure 2.37 the achieved synergistic effect is significantly greater than one produced by the ECCI. Because the position of an erasure is stored only in case of two failing cells detected in a single word, the limitations caused by the birthday paradox are removed. Therefore, more faults can be corrected in a memory block.

Table 2.14 shows the average number of faults which can be corrected with standalone implementations of Hsiao code (ECC), BR1, and ERA in the example memory ($b = 1$ and $yield = 0.5$). The column labeled *ERA-(ECC+BR1)* shows the difference between the average number of faults corrected by the ERA and the combined number of faults corrected by standalone implementations of the Hsiao code and the BR1.

As one can observe in Table 2.14 and Figure 2.38, if the memory array is divided into larger number of memory blocks then the resulting synergistic effect is better. Of course there is a limited number of faulty cells which ERA can handle. With the increased number of failing cells in the memory array the probability of having more than two faults in a single codeword also increases. This limitation can be observed in Figure 2.38.

Table 2.14: Synergy effect achieved with ERA.

# of Blocks	Average Number of Random-bit Faults (Yield = 0.5)			
	ECC	BR1	ERA	ERA-(ECC+BR1)
1	121,588	1,884	192,500	69,027
2	121,588	2,435	221,000	96,977
4	121,588	3,177	248,000	123,235
8	121,588	4,255	294,077	168,234
16	121,588	5,780	342,000	214,631
32	121,588	7,945	392,000	262,467
64	121,588	11,015	449,429	316,826

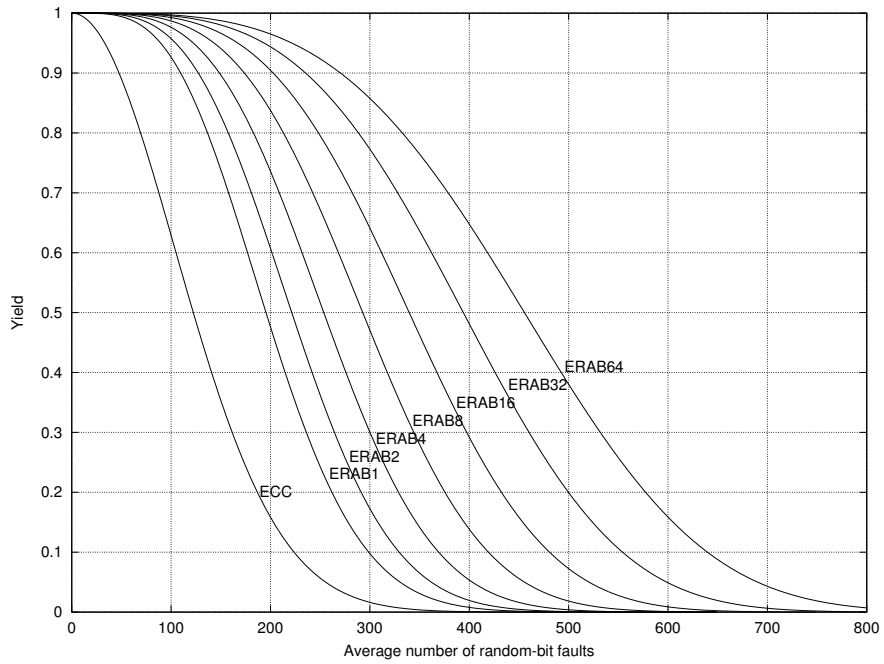


Figure 2.35: Results of yield improvement achieved with the ERA using yield formulas.

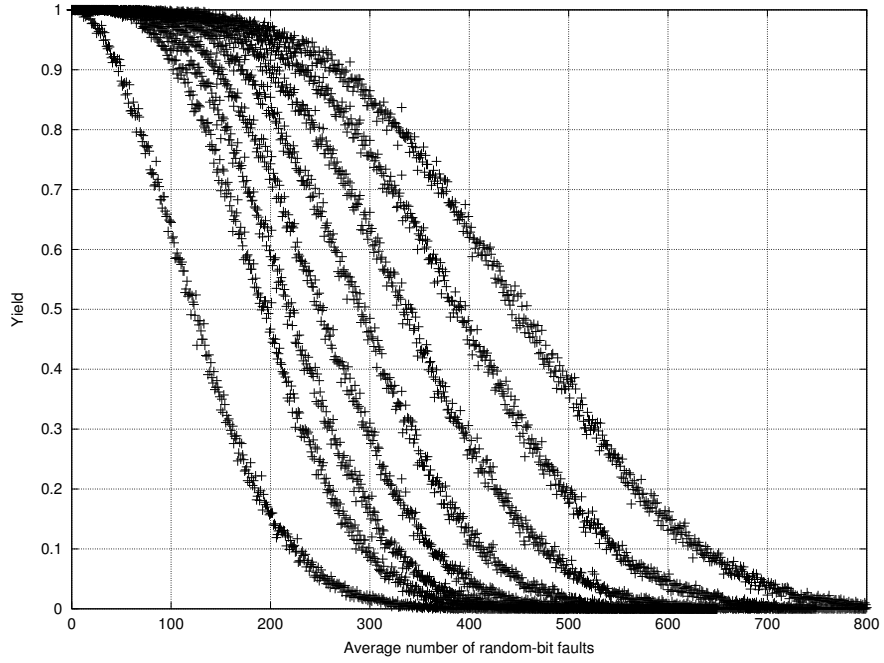


Figure 2.36: Simulation results concerning yield improvement achieved with the ERA.

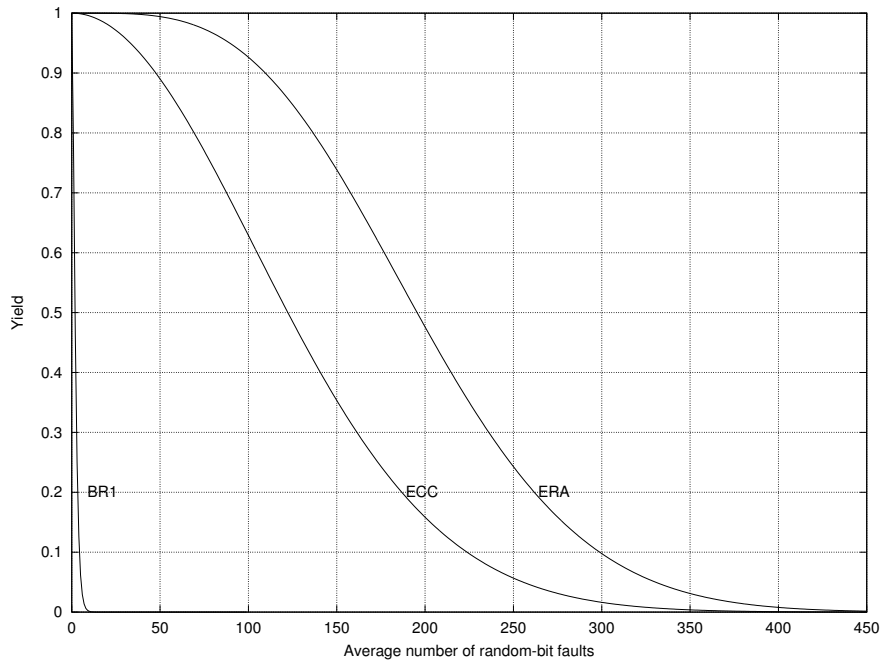


Figure 2.37: Memory yield improvement achieved with BR1, ECC, and ERA.

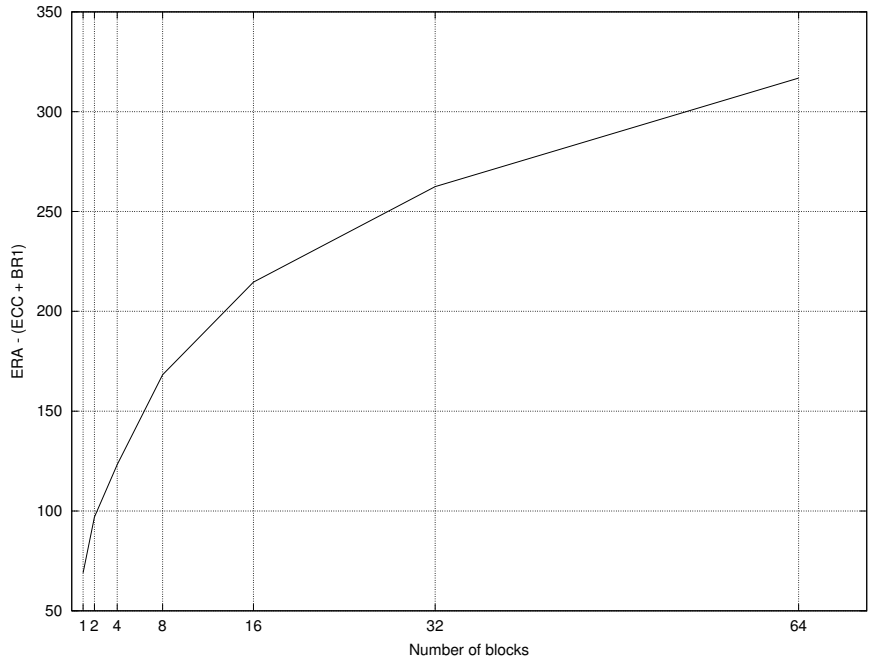


Figure 2.38: Synergistic effect achieved with the ERA.

Author's research contribution regarding SEC-DED ECCs

At first sight it may be difficult to distinguish between author's research contribution and work done by Stapper and Lee [94] and work done by Walker, Sundberg, and Black [98]. The purpose of this section is to clarify any misunderstandings and to clearly define author's research contribution regarding SEC-DED ECCs.

As previously described, in [94] Stapper and Lee showed that the so-called *birthday paradox* is limiting the repair capabilities of SEC-DED ECCs. In [94] authors also proposed a solution for circumventing limitations caused by the birthday problem by using redundant word and bit lines. That is, whenever a second failing bit occurs in a memory word protected by a SEC-DED ECC, they suggest to use either a redundant word line or bit line to replace it.

In [98] Walker, Sundberg, and Black proposed a modification to a SEC-DED ECC decoding process based on the concept of erasures. That is, with their modification it is possible to correct a single, random-bit error and a single erasure in a memory word protected by a SEC-DED ECC, simultaneously. In their approach, whenever a first error occurs in a memory block the syndrome of that error is stored. Further, when a 2-bit error occurs in a memory word located in that memory block it is assumed that the syndrome of one of errors is already known (stored syndrome). As a consequence, the 2-bit error can be corrected using approach proposed by authors in [98].

The mechanism proposed by Walker, Sundberg, and Black [98] does not take into account limitations of SEC-DED ECCs caused by the birthday paradox. The main research contribution regarding SEC-DED ECCs made by the author of this thesis is the modification of the approach proposed in [98] that takes into account problems caused by the birthday paradox. That is, instead of storing the syndrome of the first error that occurred in the memory block, the position of the second error that occurred in the memory word should be stored. To do so, however, it is required to implement a write-verify scheme in the memory controller to obtain the position of the second error in the memory word[16]. With modifications proposed by the author of this thesis not only limitations caused by the birthday problem can be circumvented but also there is no need to use redundant word or bit lines as in the approach proposed by Stapper and Lee in [94].

[16]SEC-DED ECCs cannot provide error positions when 2-bit error occurs.

2.5 Comprehensive approach

Having all the required components, i.e.:

- block-level repair for managing post-production and endurance-related faults,
- word-level repair for providing instant repair of wear-out memory cells, and
- modified ECC with increased hard-error correction capability able to correct a single-bit soft error and in the same time a single-bit hard error,

the comprehensive approach aimed at reliability improvement of existing and embedded memories can be presented.

The comprehensive approach (CA) consists of three repair mechanisms which cooperate together on different memory granularity levels (Fig. 2.39).

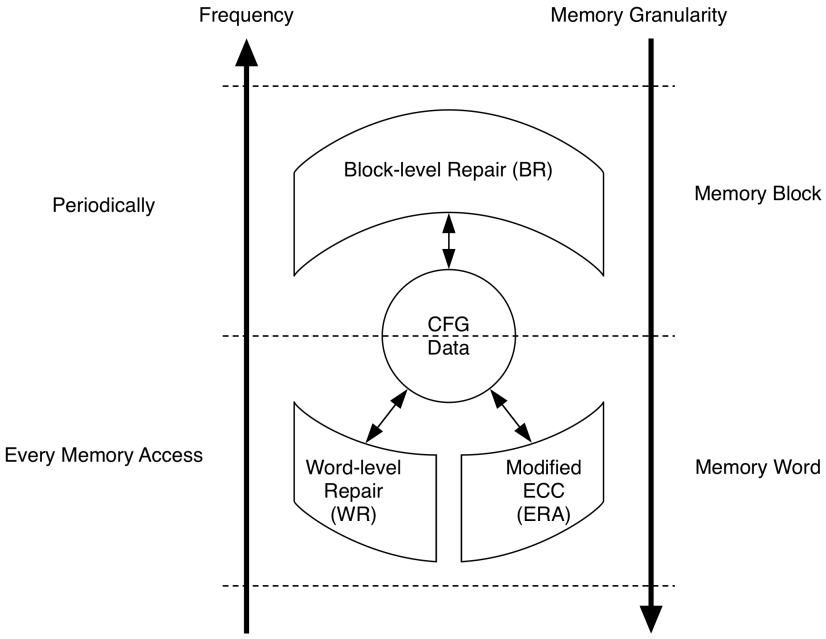


Figure 2.39: The concept of the comprehensive approach.

On the memory block level, the block-level repair is implemented. Although the block-level repair is focused mostly on post-production faults, it can also handle wear-out memory cells. The BR offers the best repair efficiency but requires the longest repair time (block rewrite procedure). As a consequence, the BR should be used mostly during post-production test and repair phases or periodically, during memory maintenance modes.

On the memory word level the word-level repair and modified ECC technique are implemented. The word-level repair provides instant repair of endurance-related faults

while the modified ECC is able to handle double-bit errors. Techniques implemented on the word level are transparent for the user and can be performed on every access to the memory.

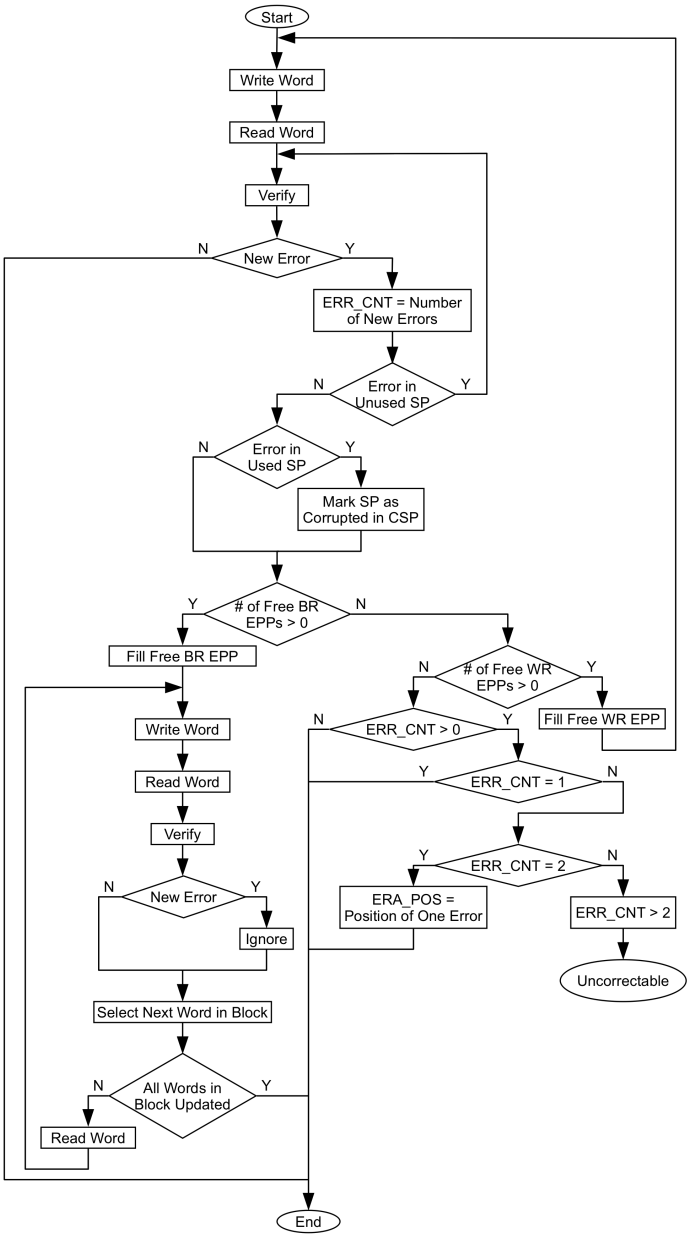


Figure 2.40: Flow chart of the comprehensive approach.

The thing that binds all repair mechanisms together is their operating range. That is, all three techniques work within a single memory block. Furthermore, all techniques are

based on managing CCSBs, either by their replacement as in BR and WR mechanisms or by correcting errors which occur in them as in ERA. As a consequence, each technique requires the same type of configuration data (positions of CCSBs).

There are many ways how WR, BR, and ERA can be combined.[17] In the thesis, the combination of the BWR and ERA, further referred as BWR_ERA or CA, will be presented and evaluated.

The flow chart presenting the working mechanism of the CA is depicted in Figure 2.40. Whenever the memory word is stored in the memory, it is read back and compared with the input data. Because the ECC is implemented in the CA, the input word consists of 32-bit user data and 7-bit parity vector. Further, the difference vector is compared with a vector of known errors. The known-errors vector stores information about positions of CCSBs obtained from active BR EPPs, WR EPPs, and ERA_POS. If there are no new errors then the stored word is assumed uncorrupted. In the other case, first, the number of new errors is counted. Next, a single error is selected from the new-errors vector and it is checked whether the single error occurred in the unused or used SP. The management of errors which occurred in unused and used SPs is the same as for BWR and will not be discussed here. Next, having the position of CCSB located in the protected area (user and ECC areas) it is checked if there are free BR SPs. If yes, then the CCSB is managed by the standard BR procedure. Otherwise, if there are free WR SPs, the CCSB is handled by the WR procedure. If there are no free BR SPs and WR SPs left, the following errors are handled by the ERA mechanism. That is, if the number of new errors is equal to 1 then it is assumed that the single-bit error will be corrected by the Hsiao code during the decoding process. If the number of new errors is equal to 2 then a single error is selected and its position is stored in the unused ERA_POS. If the ERA_POS is used, then the double-bit error cannot be handled and the memory cannot be repaired by BWR_ERA mechanism. In addition, if there are no free BR SPs, WR SPs, and the number of new errors is greater than 2, then the memory also cannot be repaired.

In the example memory there are 12 column blocks in the spare area which can be used for reliability-improving techniques. The Hsiao code requires 7 column blocks to provide SEC-DED for each memory word. As a consequence, the remaining 5 column blocks can be used for the BWR technique. With 5 spare column blocks two configurations of the BWR can be implemented: BR3WR1 and BR1WR2. Because the BR3WR1 configuration can replace greater number of CCSBs than the BR1WR2 configuration, the BR3WR1 configuration will be further used for evaluations.

As mentioned in previous section, the introduction of the ECC into a repair system severely complicates derivation of yield formulas [40]. Especially, when redundancy repair is added to the system. As a consequence, only results obtained from computer simulations are further presented. The computer simulations were performed in the similar way as simulations for BWR. The main difference between simulations performed for the BWR and BWR_ERA is the number of simulations executed for each number of random-bit faults (1000 simulations instead of 10000 like in BWR).

[17] More information on that subject can be found in Section 4.2.

In Figures 2.41 and 2.42 yield improvement achieved with different repair mechanisms is presented (b=1). Figure 2.41 shows results from the simulations while Figure 2.42 shows approximated results. As one can observe, in Figure 2.42 with BWR_ERA the best yield improvement can be achieved. Moreover, by combining BWR and ERA into a consistent system, a synergistic effect can be observed. That is, in general, the BWR_ERA can handle more random-bit faults on average than the sum of standalone implementations of BWR and ERA.

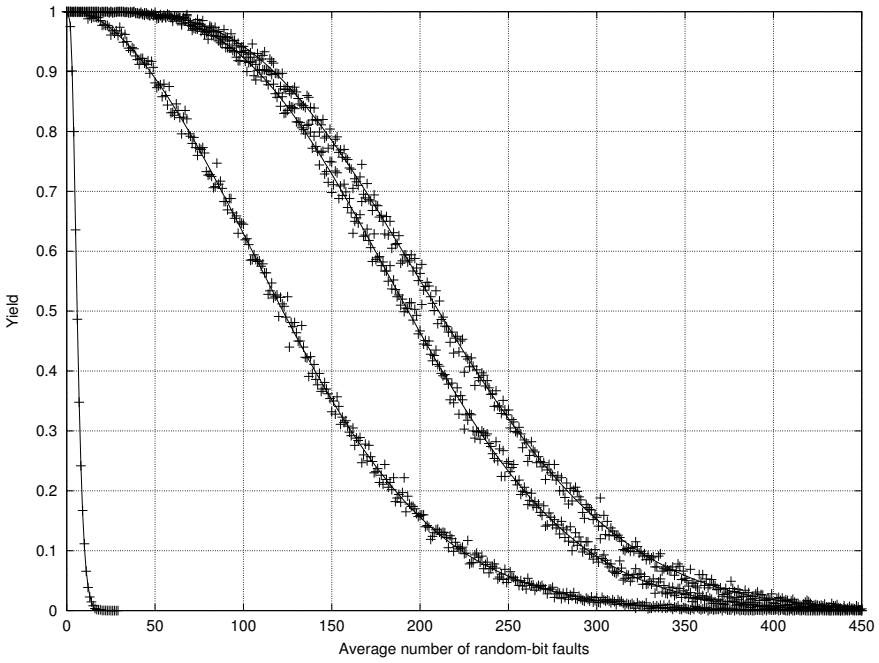


Figure 2.41: Simulation results concerning yield improvements achieved with BWR, ECC, ERA, and BWR_ERA

Table 2.15 shows the average number of faults which can be corrected with standalone implementations of BWR, ERA and BWR_ERA in the example memory (b = 1 and yield = 0.5). The column labeled BWR_ERA-(BWR+ERA) shows the difference between the average number of faults corrected by the BWR_ERA and the combined number of faults corrected by standalone implementations of the BWR and the ERA. For higher visibility, the results from Table 2.15 are also plotted in Figure 2.43. Surprisingly, in Figure 2.43 a similar effect can be observed like in evaluation of the synergy effect for the ECCI mechanism (Fig. 2.33). That is, when the memory is divided into 8 (b=8) and 16 (b=16) blocks the synergistic effect does not occur. In fact, the CA provides worse memory yield improvement than standalone implementations of the BWR and ERA. The cause of the effect is not known and requires further research. In Figures 2.44 and 2.45 implementations of the BWR_ERA and ERA for different memory block sizes are depicted.

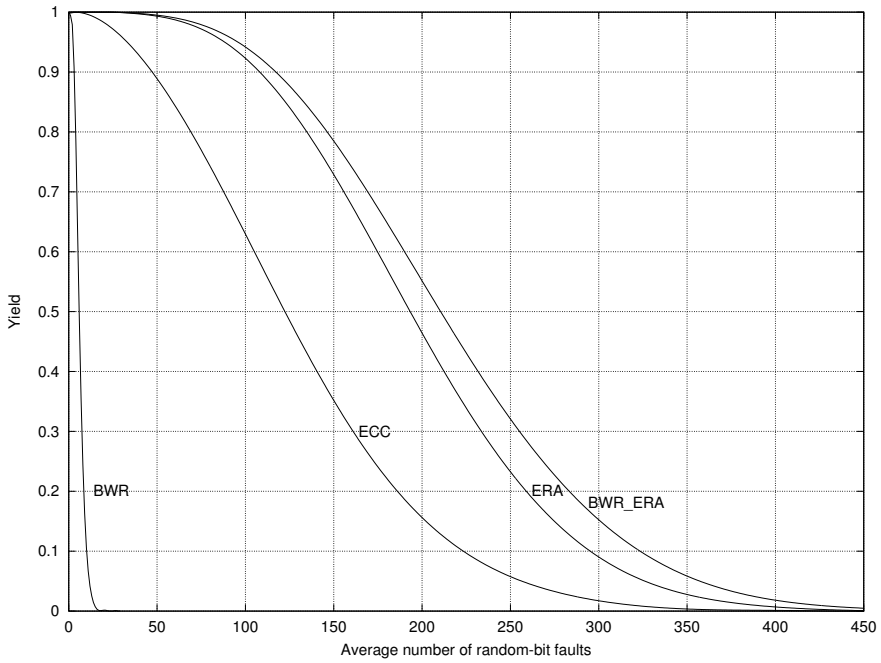


Figure 2.42: Approximated results concerning yield improvements achieved with BWR, ECC, ERA, and BWR_ERA

The main conclusion regarding presented results is that the smaller is the size of memory blocks (and thus the greater is their number) the better synergistic effect can be achieved.

As for the required memory overhead, for each memory block the BWR_ERA requires 3, 6-bit BR EPPs, 1, 6-bit WR EPP, 4-bit CSP vector, and 6-bit ERA_POS vector. The area required for different number of blocks is presented in Table 2.16.

Table 2.15: Synergistic effect achieved with BWR_ERA

# of blocks	Average number of random-bit faults (Yield = 0.5)			
	BWR	ERA	BWR_ERA	BWR_ERA-(BWR+ERA)
1	5,909	192,500	208,455	10,046
2	9,112	221,000	237,286	7,174
4	14,374	248,000	274,667	12,292
8	23,285	294,077	307,412	-9,950
16	37,746	342,000	369,000	-10,746
32	63,152	392,000	458,667	3,515
64	105,235	449,429	584,000	29,337

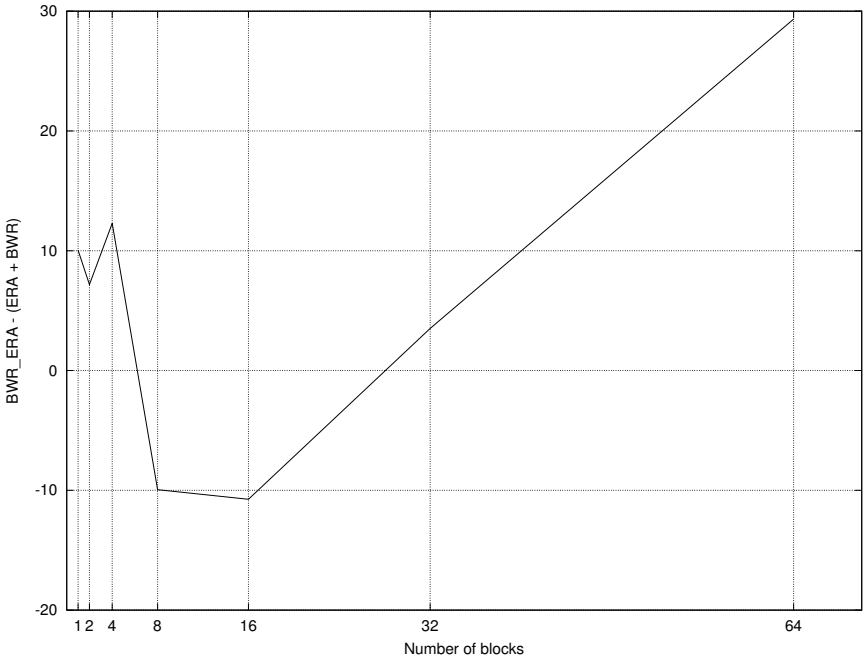


Figure 2.43: Synergistic effect achieved with the BWR_ERA

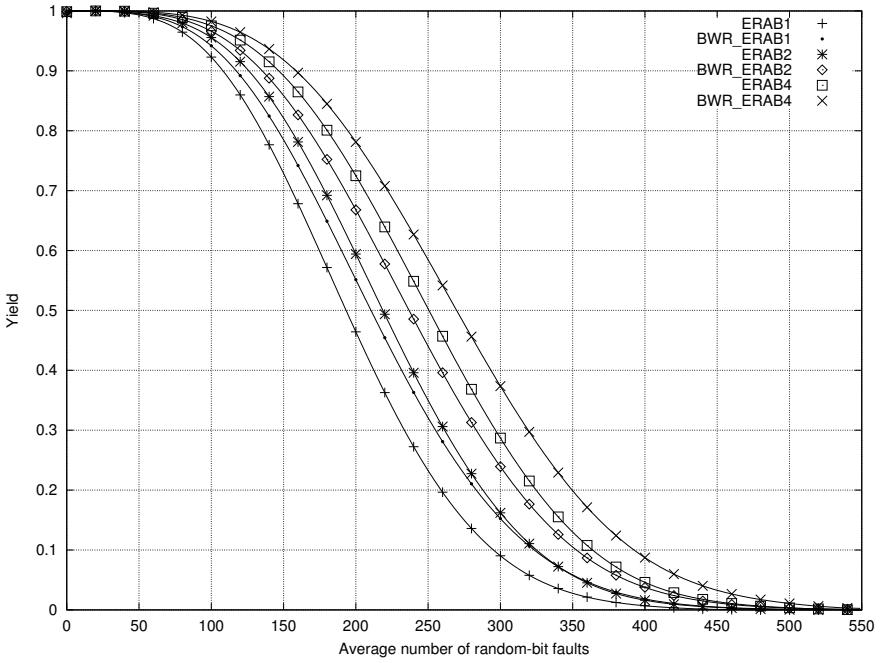


Figure 2.44: Memory yield improvement achieved with BWR_ERA and ERA techniques implemented for different number of memory blocks (1-4)

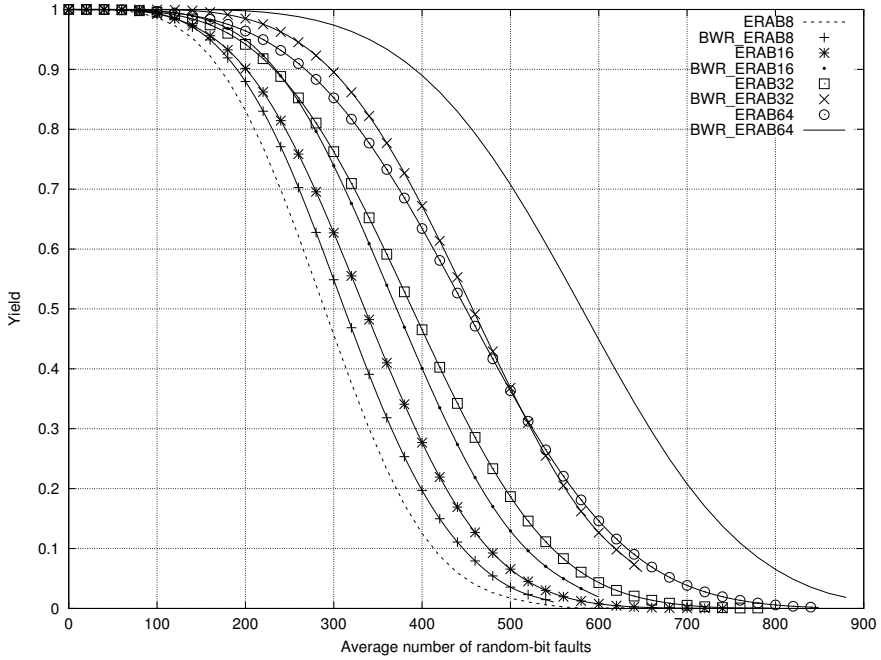


Figure 2.45: Memory yield improvement achieved with BWR_ERA and ERA techniques implemented for different number of memory blocks (8-64)

Table 2.16: Area required for the BWR_ERA implemented for different number of memory blocks.

# of Blocks	Area Overhead (bits)		Ratio (%)	
	CFG Data	ECC + SPs	CFG Data to (ECC+SPs)	(CFG Data + ECC+SPs) to Memory area
1	28	98304	0,028	27,280
2	56	98304	0,057	27,288
4	112	98304	0,114	27,304
8	224	98304	0,228	27,335
16	448	98304	0,456	27,397
32	896	98304	0,911	27,521
64	1792	98304	1,823	27,770
128	3584	98304	3,646	28,267
256	7168	98304	7,292	29,261
512	14336	98304	14,583	31,250
1024	28672	98304	29,167	35,227
2048	57344	98304	58,333	43,182
4096	114688	98304	116,667	59,091
8192	229376	98304	233,333	90,909

Chapter 3

Evaluation

Preliminary simulations of the BWR_ERA system presented in the previous chapter shows that the proposed comprehensive repair approach provides great memory reliability improvement. However, because the memory fault behavior used for simulations was based on random-bit faults modeled with the Normal distribution, the achieved reliability improvement may differ in case of different fault distribution. Moreover, performed simulations do not answer questions concerning the applicability of the BWR_ERA in a real system and its impact on system's performance. In order to address mentioned concerns the following evaluation was performed.

There were two main purposes for the evaluation. The first one was to evaluate the reliability improvement achieved with the BWR_ERA implemented for memory models which represented the fault behavior of real memory devices. The second one was to verify the applicability of the BWR_ERA for an embedded system and to evaluate its impact on system's performance.

For the evaluation concerning reliability improvement achieved with the BWR_ERA, first, appropriate memory models were generated. The memory models were based on IHP 8k x 44-bit NOR flash memories. The framework used to generate such models is further presented in Section 3.1. Next, the BWR_ERA implemented for generated models was simulated. Simulation procedure together with simulation results concerning memory reliability improvement are presented in Section 3.2.

In the second part of the evaluation, the system consisting of the BWR_ERA, generated memory models, and a microprocessor was implemented in the FPGA and emulated. In order to minimize system's performance degradation, the BWR_ERA was implemented in the form of the memory controller. The emulation system and evaluation results concerning impact of the BWR_ERA on system's performance are presented in Section 3.3.

As presented in the previous chapter, the memory reliability improvement achieved with the BWR_ERA depends on the memory block sizes for which the BWR_ERA is implemented. The smaller are memory blocks, the better is the achieved reliability improvement. On the other hand, greater number of memory blocks imposes bigger storage required for the configuration data (i.e., storage required for EPPs, CSPs, ERA_POS). Therefore, in the following evaluations the BWR_ERA was implemented for the 8k x 44 IHP NOR flash memory divided into 64 memory blocks.

3.1 Fault injection framework

Before the evaluation of the BWR_ERA could be performed, first, appropriate NVM models had to be generated. For proper evaluation, NVM models should reflect a real fault behavior of non-volatile memories. That is, faults in the NVM model should occur approximately at the same moment as in the real NVM under the same memory usage scheme. In addition, the number of faults that occurred in the NVM model should be consistent with the number of faults that occurred in the real NVM. Finally, the generated NVM model has to be usable in system simulations and emulations.

To author's knowledge, there is a lack of tools which would provide such models. The purpose of existing memory fault simulators [8], [103], [19], [78], [77] is to evaluate memory test procedures and impact of memory faults on an operating system or software. Memory models generated by existing memory fault simulators cannot by used for complete system simulations or emulations. Because of the mentioned reasons a fault injection framework was developed.

The fault injection framework facilitates the design process of a system incorporating embedded NVMs. It provides a behavior model of the embedded NVM based on expected memory usage and endurance. The generated memory model can be further used in system simulations or emulations to accurately evaluate implemented reliability-improving techniques. That is, it helps in estimating the impact of implemented techniques on system's reliability and performance.

The proposed fault injection framework consists of (Fig. 3.1):

- Memory endurance model - a model which provides information about endurance of each memory cell.
- Memory usage model - a model which generates accesses to the memory.
- Report system - a C program which simulates memory usage and reports memory errors. It uses the memory usage model for generating memory accesses and the memory endurance model for managing endurance of each memory cell.
- NVM model composed of:
  - Memory - fast memory with simple access management used for data storage.
  - Fault injection module - an HDL model which mimics modeled memory fault behavior using information provided by the report system.
- Test program - a C program which translates memory accesses from memory usage model into microprocessor's operations. In addition, the test program is used to configure the fault injection module according to the information obtained from the report system.
- Microprocessor - part of an embedded system used to execute the test program.
- FLP techniques - fault-tolerance and/or life prolonging techniques whose applicability and characteristics will be evaluated.

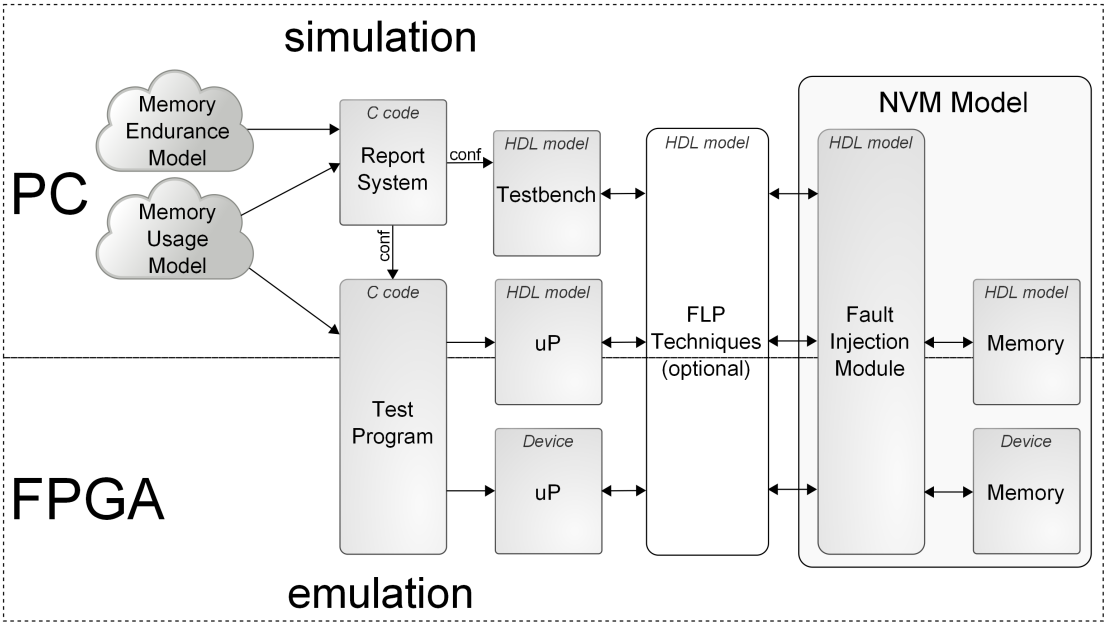


Figure 3.1: Elements of the fault injection framework. The FLP defines fault-tolerance and/or life-prolonging techniques.

Memory endurance model

The memory endurance model provides information about the lifetime of each memory cell. It has the form of a two-dimensional array which represents the layout of the modeled NVM matrix. Each array element stores a certain number of destructive memory operations which can be performed on a memory cell before it becomes damaged.

The memory endurance model should represent the endurance of modeled NVM. The endurance data should be obtained e.g. from memory endurance tests or from memory vendor. In case when accurate information about memory endurance is not available (e.g. from memory endurance tests) the lifetime of each memory cell can be modeled using e.g. probability distributions (if applicable).

Memory usage model

The memory usage model reflects the expected memory usage scheme. It is represented by a C function containing algorithms which generate appropriate memory accesses. The accuracy of the NVM model strictly depends on the accuracy of the memory usage model.

The memory usage model should be able to generate memory data patterns and addressing schemes which will be used in a real system. For example, if modeled NVM will be used for storing program code, the instruction set of a microprocessor should be used as a data pattern. When modeled NVM will be used for storing data from external sensors then data patterns which can be obtained from external sensors should be generated in the memory usage model. In case when reliability-improving techniques

are incorporated into the system, their models have to be included in the memory usage model.

If the NVM model will be also used for emulation purposes it is crucial that the algorithms included in the memory usage model can be executed on system's microprocessor. The selection of algorithms generating memory accesses should be done carefully. Poor implementation can lead to longer simulation and emulation times or inability to perform emulation. This concerns especially random number generators if used.

Report system

The most important part of the fault injection framework is the report system. The report system performs memory usage simulation according to the memory usage model and the memory endurance model. Before the simulation can be performed, the report system has to be properly configured, i.e.:

- memory parameters have to be defined, i.e.: number of address and data bits, number of memory rows and columns;
- functions for logical to topological address mapping have to be implemented;
- destructive operations have to be selected;
- fault models[1] for damaged cells have to be selected;
- simulation stop condition has to be defined;

Next, when the report system is properly configured, the simulation can be executed. The report system simulates memory usage and records occurrences of memory errors. After reaching the number of error occurrences defined in the configuration (simulation stop condition), the report system stops the simulation. Next, a general report is created. The general report stores chronological information about all error occurrences in memory words. Each record of general report looks as follows:

$$[acc\_nr][addr][pos1][type1][en1]...[posN][typeN][enN]$$

Where acc_nr and $addr$ fields give information about when (which memory access) and where (which memory word) errors occurred. The rest of the record holds information about all errors in the memory word which occurred during whole simulation, and which of these errors appeared at acc_nr memory access. That is, for each error in the memory word, the record holds information about position of the error - $posX$, its type - $typeX$ and information if error occurred (is enabled) at acc_nr memory access - enX. To clarify, below is an example presenting part of the general report:

$$[acc\_nr]\ \ [addr]\ [pos1]\ [type1]\ [en1]\ [pos2]\ [type2]\ [en2]$$

[1] The presented version of the report system implements stuck-at fault model only.

[1200]	[0]	[1]	[2]	[0]	[2]	[2]	[1]
[1210]	[1]	[4]	[1]	[1]	[0]	[0]	[0]
[1500]	[0]	[1]	[2]	[1]	[2]	[2]	[1]

First record shows that the first error in the memory appeared at 1200^{th} memory access at address 0. It also shows that memory word at address 0 has overall 2 errors in the simulation: one stuck-at-1[2] error at 1^{st} bit position and second stuck-at-1 error at 2^{nd} bit position. In the first record the enable bit is set only for the error located at 2^{nd} bit position. This indicates that at 1200^{th} memory access only stuck-at-1 error located at 2^{nd} bit position occurred in the memory word located at address 0. The second record shows that at 1210^{th} memory access at address 1 a stuck-at-0 error occurred at the 4^{th} bit position. Finally, the 3^{rd} record shows that the second stuck-at-1 error in the 0^{th} memory word occurred at the 1500^{th} memory access.

The general report created by the report system represents the memory fault behavior strictly related to expected memory usage and endurance. In case of different memory usage or different expectations about memory endurance the memory usage simulation has to be re-executed in order to generate a new general report. Based on the general report, the report system creates appropriate configuration files for the fault injection module and for the test program. These configuration files are required to properly configure the NVM model.

NVM Model

The NVM model is a module which is able to mimic the fault behavior of the selected NVM. The properly configured NVM model can be used for simulation or emulation purposes. It consists of the memory module and the fault injection module (Fig. 3.2).

Memory - The NVM model utilizes SRAM as a storage medium. The main benefit of using this type of memory is that SRAM has unlimited endurance thus it will not influence the behavior of the NVM model. Moreover, it is a well-known memory which means that there are a lot of HDL models and devices which can be used for simulation and emulation. The SRAM can be found on most FPGA devices which in some cases can facilitate the implementation of the NVM model. Finally, the SRAM is simpler and faster than most of NVMs, which leads to faster simulation and emulation speeds.

Fault injection module - The fault injection module is responsible for modeling memory fault behavior. It uses information provided by the report system to inject previously simulated memory errors at specified memory accesses. It consists of (Fig. 3.2):

- FIFO - First-in first-out memory which stores chronological information about memory accesses at which errors occurred. Each FIFO word contains access num-

[2] In the Report System, stuck-at-1 error is represented by number 2, stuck-at-0 by number 1, and memory cell without errors by number 0.

ber, memory address, and error enable vector. The error enable vector defines which errors in a word should be activated.

- mem_ctrl bus - memory control signals. Changing control signals determine different accesses to the memory and are used to increase internal access counter.
- SRAM - stores information about all errors occurred in corrupted memory words. Each SRAM word contains positions and types of errors and error enable vector.
- CAM - Content addressable memory is used to map input address provided to the NVM model (mem_addr) into SRAM address. The CAM stores all memory addresses at which errors occurred. In case when memory input address (mem_addr) matches address stored in the CAM, a CAM hit signal is set and the address of corresponding SRAM word is driven to CAM output.
- mem_addr bus - memory address used for CAM and for verification purposes[3].
- Fault injection controller - manages all internal memories, keeps track of memory accesses, and performs error injection.
- Registers - accessed by AMBA APB bus are used for configuration purposes and provide access to FIFO, CAM, and SRAM.
- mem_do bus - output data from the memory.
- mem_do_err bus - output data from the memory with injected faults.

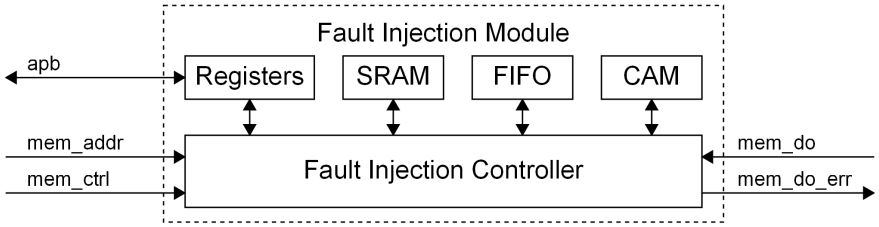


Figure 3.2: Block diagram of the fault injection module.

Before simulations or emulations with the NVM model can be performed, the fault injection module has to be initialized. That is, information provided by the report system have to be loaded into appropriate memories, i.e., FIFO, SRAM, and CAM. After initialization, the fault injection module is ready to perform error injection. The fault injection controller keeps track of memory accesses and memory input address. Each time when memory control signals change into memory write or read control signals, an internal access counter is incremented (Fig. 3.3). In parallel, the memory input address

[3]That is, whenever an error should be injected, it is checked if the mem_addr matches the memory address from the FIFO word. In case of a mismatch appropriate bit is set in the status register indicating that the fault injection was not properly performed.

is compared with the content stored in the CAM. In case of a match, the CAM sets hit signal and drives the appropriate SRAM address to the output. Next, the SRAM word addressed by CAM is read. Information from SRAM word is then used to inject errors in the memory output word. That is, error positions whose enable bit is set are used to change the original bits in the memory output word according to the specified error type. In case when the access counter matches the access number from FIFO word (Fig. 3.4), the error enable vector from the FIFO is used instead of the error enable vector from SRAM to activate appropriate errors in the word. Moreover, the fault injection controller issues a write operation to the SRAM to update the error enable vector. In the same time another word from the FIFO is read. Because of SRAM update operation, whenever memory access to the corrupted memory word will be issued, errors will be injected according to the updated error enable vector in SRAM.

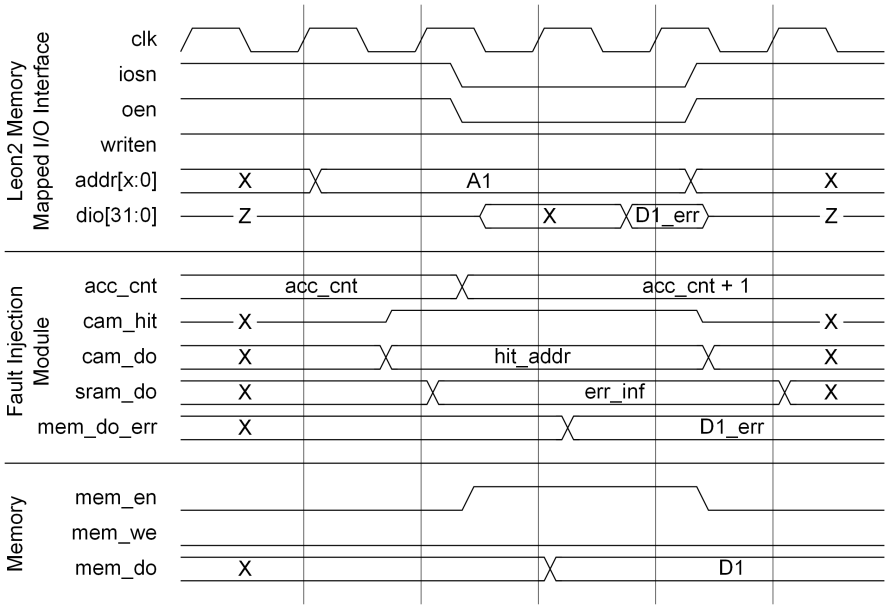


Figure 3.3: Behavior of the NVM model connected to the Leon2 processor HDL model as a memory mapped I/O device. Figure presents situation when during memory read access issued by Leon2 memory controller information from SRAM is used to inject errors in the memory output word.

Framework usage scenarios

The fault injection framework consists of several simple modules which can be combined in different ways. Based on the combination, various simulation and emulation scenarios can be achieved such as those presented below:

Evaluating the applicability of the selected NVM memory - Using the report system together with the memory endurance model and the memory usage model the applica-

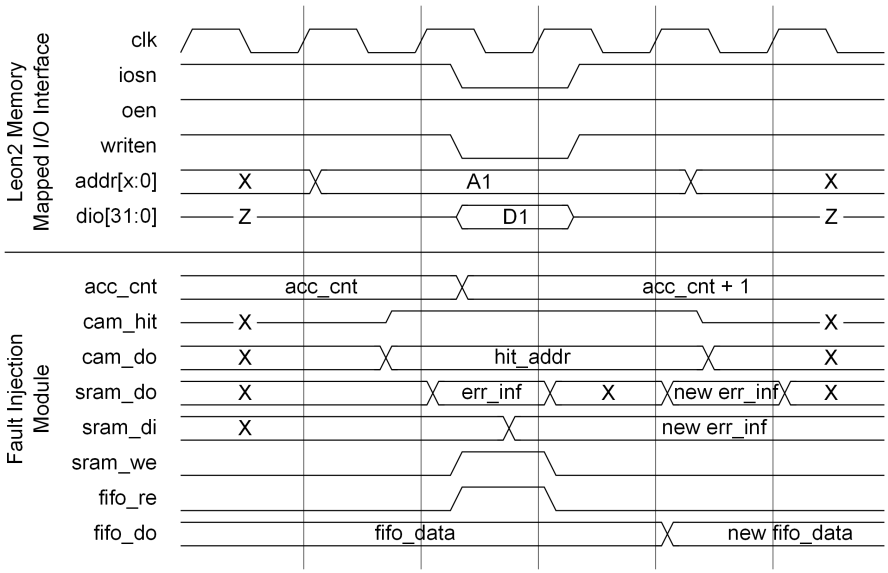


Figure 3.4: Behavior of the NVM model connected to the Leon2 processor HDL model as a memory mapped I/O device. The figure presents the situation when during memory write access issued by the Leon2 memory controller the value of access counter matches the access number stored in FIFO.

bility of selected NVM for a target system can be easily evaluated. The memory usage simulation performed by the report system can give information about potential problems with the modeled memory. This information can be further used to decide whether reliability-improving techniques are required or different memory technology should be used.

Evaluating reliability-improving techniques - The Report System gives the possibility to check whether the selected repair mechanisms are sufficient to increase the reliability and/or lifetime of the modeled NVM. Different repair techniques can be implemented in the memory usage model and simulated with the report system. Based on the outcome, reliability-improving techniques offering the best results can be selected for further implementation.

Verifying and simulating HDL models of reliability-improving techniques - Repair mechanisms implemented in the memory usage model can be used as a golden model in verifying their implementation in the HDL. In order to do so, first, the memory usage model incorporating reliability-improving techniques should be used for memory usage simulation. Memory traces generated by the report system will be further used for verification. Next, the memory usage model only with algorithms generating memory accesses should be used as memory stimuli for the HDL testbench. In HDL simulation, the output from the NVM model should be compared with previously simulated memory traces.

Any discrepancies in comparison will indicate problems with the HDL implementation of repair techniques.

Evaluating and simulating complete system - The NVM model provided by the fault injection framework can be used for a complete system simulation. By doing this, the impact of the memory fault behavior on whole system (with or without repair techniques) can be evaluated. In addition, the impact of reliability-improving techniques on a system's performance can be verified.

Evaluating and emulating complete system - In case of large embedded systems a whole system simulation can take days and may be unpractical. In such situations a better approach would be to perform system emulation. For the price of observability one would gain faster evaluation of the system. The NVM model provided by the fault injection framework can be used in system emulation.

Fault injection framework evaluation

The evaluation of the fault injection framework consisted of two schemes. The main purpose of the first evaluation scheme presented in [87] was to validate the correctness of the fault injection framework and its applicability for system simulation and emulation purposes. The second evaluation scheme presented in [88] was focused on evaluating the system's performance degradation caused by implementation of the NVM model. Further, both evaluation schemes and obtained results are presented.

First evaluation scheme

The first evaluation scheme consisted of two parts. In the first part, the report system was used to perform memory usage simulations with different memory endurance and usage models and with different parameters of the modeled memory. The purpose of the first evaluation was to determine memory usage simulation speeds and sizes required for memories in the fault injection module. In the second part, selected results from the first part were used to create NVM models which were used in system emulations in the FPGA. The purpose of the second part was to evaluate correctness of the proposed approach and to determine emulation speeds.

Report system

For this evaluation the report system was used to perform memory usage simulations for 8kB, 16kB, 32kB, and 64kB flash memory models[4]. For each memory size the report system used three different memory endurance models generated with Normal distribution with different cells' mean lifetime and standard deviation values:[5]

[4]This implies that write and erase operations are destructive, i.e. they decrease endurance of memory cells. Moreover, before new data can be stored in the memory, first, memory has to be erased.

[5]The use of Normal distribution for modeling memory's endurance is a common practice for evaluating reliability-improving techniques [84][46][75].

- 1e5_1e4 model - mean = 10^5, std. deviation = 10^4;
- 1e4_1e3 model - mean = 10^4, std. deviation = 10^3;
- 1e3_1e2 model - mean = 10^3, std. deviation = 10^2;

and two memory usage models:

- uniform - In the uniform model memory accesses begin from 0^{th} memory address. Random data is stored in the memory and read back. Next, address is incremented and another random data is stored and read back. When whole memory is written, the erase operation is performed, and whole procedure repeated.
- pareto - In the pareto, similarly like in the uniform model, randomly generated data is stored and read back. The difference is that in the pareto model memory matrix is divided into two areas: one containing 20% ($area1$) and the other 80% ($area2$) of memory words. Whenever any of the areas is completely written the whole memory is erased and the next area where data will be stored is randomly selected. The $area1$ has 80% chances while the $area2$ has 20% chances of being selected for data storage.

Each configuration was simulated until 1%, 5%, and 10% of corrupted bits were reported. Moreover, the report system started recording memory accesses when the endurance of any memory cell was equal to 1. Memory usage simulations were performed on Intel Core i5 M560 @ 2.67GHz, 3GB RAM machine with 32-bit Windows 7 OS. The report system does not use parallelism therefore, during simulations, only one core was used for computations. For generating random numbers MWC generator developed by George Marsaglia was used [65]. The MWC generator was selected because it is fast, provides period $\sim 2^{60}$ and can be easily implemented in embedded systems. Each configuration was simulated 10 times with different seeds for the random number generator.

Average results from the evaluation are gathered in Table 3.1 and Table 3.2. In each table the *Access Counter* shows the average number of accesses to the memory before the required percentage of errors was reported. *Execution Time* reports the average time for executing whole memory usage simulation. *Memory Ratio* field is the ratio of total memory size required for fault injection module to the size of the modeled memory.

As one can observe, simulation time strictly depends on the memory endurance model.[6] That is, the higher is the endurance of modeled memory the more memory operations are required to wear-out memory cells.

Other factors which have impact on the simulation time are memory size and number of errors to report. The latter factor has also the biggest influence on the total memory size required for SRAM, CAM, and FIFO. Larger number of errors requires bigger memory for error-related information. In addition, it can be seen that the lower is the endurance

[6] Unfortunately, because of the OS, the memory usage simulation time performed by the report system was affected by other processes (e.g. look at results for 32kB, 1e5_1e4, uniform configurations). Nevertheless, general dependencies concerning simulation time can be observed. Other simulation parameters were not affected and are accurate and deterministic.

Table 3.1: Average results for configurations using uniform memory usage model.

	Memory size	**64kB**			**32kB**			**16kB**			**8kB**		
	% of errors	**10**	**5**	**1**	**10**	**5**	**1**	**10**	**5**	**1**	**10**	**5**	**1**
1e5_1e4	**Acc. cnt. [k]**	833107,7	706386,0	573425,9	384553,5	342529,5	264869,4	178850,5	165499,9	107278,2	88028,3	82877,8	48348,3
	Exec. time [s]	1491,3	1431,8	1321,7	1105,1	1471,8	1204,9	700,9	764,6	701,4	398,5	329,0	258,6
	Memory ratio	1,55	0,71	0,15	1,48	0,66	0,14	1,42	0,64	0,13	1,30	0,60	0,13
1e4_1e3	**Acc. cnt. [k]**	85104,4	70987,0	55639,8	39180,4	32918,3	25738,4	19445,7	16257,0	11405,0	9488,6	7696,8	5433,2
	Exec. time [s]	146,4	141,2	130,7	94,3	77,8	61,9	78,5	75,7	70,9	28,8	28,9	26,1
	Memory ratio	1,45	0,67	0,14	1,40	0,63	0,13	1,31	0,62	0,13	1,26	0,58	0,12
1e3_1e2	**Acc. cnt. [k]**	8174,6	8163,4	5725,0	3961,3	3555,4	2581,0	1710,7	1685,4	1169,0	838,1	795,0	531,6
	Exec. time [s]	14,9	14,2	13,1	7,0	6,7	6,0	7,9	7,5	7,0	3,1	2,9	2,8
	Memory ratio	1,50	0,65	0,14	1,36	0,61	0,13	1,25	0,57	0,12	1,16	0,53	0,12

Table 3.2: Average results for configurations using pareto memory usage model.

	Memory size	**64kB**			**32kB**			**16kB**			**8kB**		
	% of errors	**10**	**5**	**1**	**10**	**5**	**1**	**10**	**5**	**1**	**10**	**5**	**1**
1e5_1e4	**Acc. cnt. [k]**	375109,9	348186,4	236882,0	184572,1	172192,9	109174,4	87923,6	73777,8	53283,6	39234,4	36116,7	24793,1
	Exec. time [s]	1942,4	1594,7	1166,0	970,5	925,8	841,9	388,3	332,3	292,2	184,3	202,1	162,0
	Memory ratio	1,43	0,81	0,17	1,35	0,78	0,15	1,28	0,73	0,14	1,23	0,69	0,13
1e4_1e3	**Acc. cnt. [k]**	39961,3	33575,2	24975,6	19286,8	16631,2	11816,8	9235,7	7664,9	5022,1	4405,5	3105,6	2604,9
	Exec. time [s]	159,1	219,7	151,1	94,5	91,1	83,4	32,6	33,7	32,6	17,8	18,5	15,4
	Memory ratio	1,38	0,82	0,16	1,32	0,77	0,15	1,25	0,70	0,14	1,19	0,67	0,13
1e3_1e2	**Acc. cnt. [k]**	3497,7	3493,8	2374,1	1772,2	1547,5	1180,7	812,4	745,8	530,7	367,0	340,8	244,7
	Exec. time [s]	16,6	15,1	13,7	9,8	8,9	8,4	3,8	3,2	2,9	1,8	1,7	1,6
	Memory ratio	1,30	0,81	0,15	1,26	0,74	0,15	1,18	0,67	0,13	1,13	0,63	0,12

of the memory the smaller is the total memory size required by fault injection module. This is caused by lower number of bits required for the FIFO where memory access numbers of error occurrences are stored.

As for the memory usage model, the number of memory accesses before the given percentage of errors was reported is lower for the pareto model than for the uniform model. This is caused by the memory usage pattern. In the pareto model 80% of memory accesses are located in the memory area containing 20% ($area1$) of memory words. As a result, memory cells located in $area1$ weared-out faster than others.

System emulation

In this part, four results from the first evaluation were used to generate four NVM models:

- 1^{st} model - 64kB flash memory, uniform usage scheme;
- 2^{nd} model - 64kB flash memory, pareto usage scheme;
- 3^{rd} model - 32kB flash memory, uniform usage scheme;
- 4^{th} model - 32kB flash memory, pareto usage scheme;

Next, four embedded systems were created. Each system consisted of a single NVM model and the Leon2 processor HDL model. Resulting systems were implemented in the GR-CPCI-XC2V Virtex2 FPGA board [73] and emulated. For each system a different test program was used during emulation. The test program was responsible for:

- initializing memories in the fault injection module with appropriate data generated by the report system,
- performing accesses to the memory according to the memory usage model, and
- verifying the correctness of the model.[7]

After each task performed by the test program, the value of the on-chip 64-bit timer was read and recorded. Parameters of selected configurations and results from system emulation are reported in Table 3.3. Table 3.3 presents only those configurations which could fit together with Leon2 processor HDL model in the Virtex2 FPGA. The biggest problem during the system synthesis was the CAM from fault injection module. Although FIFO, SRAM, and memory for NVM's model data storage were implemented using FPGA's SRAM resources, the CAM was implemented with registers. Because of that not all configurations from the first part of the evaluation could be used for system emulation and FPGA board's clock had to be reduced to 12MHz. In Table 3.3 *Initialization time* shows the time required to load configuration data into memories in the fault injection module. *Execution time* is the time required to complete memory usage emulation. *Verification time* is the time required to check if errors were correctly injected.

Table 3.3: System emulation results for selected configurations.

Memory Size	64kB		32kB	
Mem. Usage Model	Uniform	Pareto	Uniform	Pareto
% of Errors	1	5	5	10
Mem. End. Model	1e5_1e4	1e5_1e4	1e3_1e2	1e4_1e3
Acc. Cnt. [k]	587843,1	386378,1	2854,2	19359,8
Memory Ratio	0,17	0,63	0,73	1,48
Init. Time [s]	0,0050	0,0246	0,0152	0,0257
Exec. Time [s]	1347,2096	966,1335	6,6603	48,4151
Verifi. Time [s]	9,6565	30,9700	11,4920	11,5108

Note that the NVM used SRAM memory as a data storage. As a result SRAM access times were used for the NVM model.

As expected, the execution time strictly depends on the number of memory accesses which were required to report appropriate percentage of errors. The bigger is the number of memory accesses the longer is the execution time. As shown in Table 3.3 the initialization time is very small in comparison to execution and verification times and is strongly related to the number of memory errors (i.e. the size and structure of configuration data). The verification time depends on the modeled memory size and also on the structure of configuration data, where the letter factor has the biggest impact. The bigger and more complex is the structure of configuration data the bigger is data manipulation effort for Leon2 processor.

During system emulations no errors were reported during verification phase performed by the test program. Furthermore, no injection errors were reported in the status register. All errors in all systems were correctly injected.

Second evaluation scheme

The main purpose of the second evaluation scheme was to determine system performance degradation when the NVM model is used. In order to do so, first, 30 different 32kB NOR flash memory models were generated. For each memory model the memory endurance model was generated with Normal distribution with mean = 10^4 and standard deviation = 10^3. Moreover, for each memory model the same memory usage model - pareto model - was used. All 30 NVM Models were generated according to the number of error events to simulate/emulate, i.e.:

- 1p group - 10 NVM Models which represented 32kB NOR flash memories with 1% of error events.

[7] In the properly emulated system, errors in memory words should reflect error information in the configuration data stored in the Fault Injection Module.

- 5p group - 10 NVM Models with 5% of error events.
- 10p group - 10 NVM Models with 10% of error events.

For each NVM Model, MWC random number generator [65] with different seeds was used to generate memory endurance model and memory usage model.

All NVM Models were emulated together with Leon2 processor HDL model in GR-CPCI-XC2V Virtex2 FPGA board. Results regarding generation and emulation of NVM Models are further presented.

Report system

The report system was utilized to simulate the memory usage using memory endurance models and memory usage models. Next, generated reports were used to create appropriate configuration files: input data for fault injection module memories (CAM, SRAM, and FIFO), and data for test program. Memory usage simulations were performed on Intel Core i5-4250U, 4GB RAM machine.

Results from memory usage simulations performed by the report system are gathered in Tables 3.5, 3.6, and 3.7. In each table the *Acc. cnt* shows the number of accesses to the memory before the required percentage of error events was reported, *Exec. time* reports time for executing whole memory usage simulation, and *Memory ratio* field is the ratio of total memory size required for fault injection module to the size of modeled memory (32kB NOR flash memory).

As expected, the number of accesses to the memory, before required percentage of error events was reported, increases with the number of error events. Similar observation applies to the execution time of the simulation with one exception. Because of used memory usage model and memory endurance model, after reporting specified number of error events the number of destructive accesses required to create more faults diminishes. As a result, there is a small difference between simulation execution times for 5p and 10p groups. Higher standard deviation and/or different memory usage model would increase the difference in simulation execution times. Finally, the memory required by fault injection module strictly depends on the number of memory faults and their occurrence (memory address and position within word). On average, for simulated models, the fault injection module requires additional 15.6%, 78.7%, and 133.6% of modeled 32kB memory for 1p, 5p, and 10p groups, respectively.

System emulation

From each group (1p, 5p, and 10p) one NVM Model which required the biggest amount of resources was synthesized together with Leon2 processor HDL model and implemented in the FPGA. By doing this, only three synthesized systems were required for evaluation purposes. Other NVM Models which required less resources were emulated in synthesized systems. Furthermore, a fourth synthesized system - BARE system - consisting of Leon2

processor HDL model and fault-free 32kB NOR flash memory model[8] was used as a reference.

The most important synthesis results are gathered in Table 3.4. In Table 3.4 *RAMB16s* defines used FPGA's SRAM resources, *SLICEs* defines used resources of FPGA's configurable logic blocks, and *Max. freq.* defines the maximum frequency for synthesized systems. The biggest problem during system synthesis was the CAM from fault injection module. Although FIFO, SRAM, and storage memory for NVM model were implemented using FPGA's SRAM resources, the CAM was implemented mostly with registers. As a result, the bigger was amount of injected faults, the bigger CAM was required by fault injection module and the bigger and slower was synthesized system. One of the solutions to this problem would be to provide external CAM to store addresses of corrupted memory words. Another solution would be to implement CAM in a FPGA-friendly way using internal SRAM's. Unfortunately, this would exploit all FPGA's SRAM resources and there would be no memory for SRAMs and FIFO memories required by the NVM model.

Table 3.4: Synthesis results.

	BARE	MAX_1p	MAX_5p	MAX_10p
RAMB16s	68,056%	70,139%	77,778%	81,944%
SLICEs	19,229%	40,921%	99,994%	99,994%
Max. freq. [MHz]	27,121	23,165	16,032	15,893

In the next part of the evaluation all generated NVM models were emulated in the FPGA using synthesized systems and 12MHz system clock. For each NVM model the same test program was used during emulation. The test program was responsible for:

- initializing memories in the fault injection module with appropriate data generated by the report system. For each NVM model different initialization data was used,
- performing accesses to the memory according to the memory usage model, and
- verifying the correctness of the model.

For the BARE system, the test program performed only accesses to the memory according to memory usage model. In the BARE system, no fault injection was performed.

After each task executed by the test program, the value of the on-chip 64-bit timer was read and recorded. Only results from memory usage emulations are reported in Tables 3.5, 3.6, and 3.7. In Tables 3.5, 3.6, and 3.7 *Fault inj. time* is the time required to complete memory usage emulation for NVM models from 1p, 5p, and 10p groups, and *Sim. time* is the time required to complete memory usage emulation in the BARE system.

[8]NVM Model without the fault injection module.

Table 3.5: Simulation and emulation results for NVM Models from 1p group.

NVM Model	1	2	3	4	5	6	7	8	9	10	MIN	AVG	MAX
Acc. cnt [k]	48280,02	48693,53	49680,95	48843,59	48688,38	49076,68	49200,16	49334,54	49361,13	48531,39	48280,02	48969,04	49680,95
Exec. time [s]	26,74	28,90	29,77	29,64	30,21	25,23	29,64	25,62	29,15	28,55	25,23	28,34	30,21
Memory ratio	0,154	0,154	0,154	0,154	0,154	0,154	0,154	0,154	0,154	0,172	0,154	0,156	0,172
Fault inj. time [s]:	151,14	152,52	155,58	152,92	152,45	153,67	154,12	154,49	154,56	151,97	151,14	153,34	155,58
Sim. time [s]:	150,09	151,41	154,45	151,84	151,37	152,62	152,98	153,41	153,47	150,89	150,09	152,25	154,45

Table 3.6: Simulation and emulation results for NVM Models from 5p group.

NVM Model	1	2	3	4	5	6	7	8	9	10	MIN	AVG	MAX
Acc. cnt [k]	53686,06	53065,89	53359,48	53759,45	53869,63	52814,92	53262,53	52542,79	53547,29	53180,36	52542,79	53308,84	53869,63
Exec. time [s]	32,25	32,96	28,93	29,55	29,50	31,01	30,07	31,72	28,54	29,67	28,54	30,42	32,96
Memory ratio	0,969	0,629	0,758	0,758	0,758	0,758	0,828	0,828	0,828	0,758	0,629	0,787	0,969
Fault inj. time [s]:	168,08	166,15	167,03	168,30	168,66	165,33	166,74	164,49	167,62	166,49	164,49	166,89	168,66
Sim. time [s]:	166,90	164,96	165,88	167,15	167,48	164,17	165,58	163,32	166,49	165,32	163,32	165,73	167,48

Table 3.7: Simulation and emulation results for NVM Models from 10p group.

NVM Model	1	2	3	4	5	6	7	8	9	10	MIN	AVG	MAX
Acc. cnt [k]	55371,10	54893,95	54229,42	55525,91	56044,18	56651,64	54851,30	54812,63	56320,05	53937,08	53937,08	55263,73	56651,64
Exec. time [s]	28,94	29,15	31,36	29,88	32,82	32,45	29,15	29,76	31,21	29,99	28,94	30,47	32,82
Memory ratio	1,414	1,336	1,258	1,492	1,336	1,180	1,414	1,336	1,258	1,336	1,180	1,336	1,492
Fault inj. time [s]:	173,33	171,82	169,72	173,84	175,45	177,38	171,70	171,55	176,32	168,81	168,81	172,99	177,38
Sim. time [s]:	172,13	170,61	168,58	172,60	174,23	176,14	170,52	170,38	175,09	167,65	167,65	171,79	176,14

As expected, the fault injection time depends on the number of memory accesses which were required to report appropriate percentage of error events. The bigger is the number of memory accesses the longer is the execution time. Similarly like in memory usage simulations performed by the report system, the difference between fault injection times for NVM models from 5p and 10p groups is smaller then between NVM models from 1p and 5p groups. The difference between times of memory usage emulations for NVM models from 1p, 5p, and 10p groups and fault-free NVM model from BARE system is negligible (less than 0.8%). In author's opinion, the main cause of the difference lies in the test program and the compiler. Although the test program for BARE system used the same memory usage model, it was devoid of initialization and verification procedures.

During emulations, there were no errors during the verification phase. Moreover, no errors were reported in the status register. All faults in all systems were correctly injected.

3.2 BWR_ERA simulation

In order to evaluate the memory reliability improvement achieved with the BWR_ERA, first, proper NVM models had to be developed. That is, the NVM fault behavior based on expected memory endurance and usage pattern had to be generated. As described in the previous section, the task of generating configuration data for NVM models is performed by the report system. Before memory usage simulations with the report system can be executed, appropriate memory endurance models and the memory usage model have to be developed.

Memory endurance models

The memory endurance models were created according to test results obtained for IHP 8k x 44-bit NOR flash memories. Seventy-nine IHP 8k x 44-bit NOR flash memory chips were produced using IHP 250 nm fabrication process. Next, all memory chips were tested on the Verigy Advantest V93000 SoC test system [1]. Table 3.8 presents the test procedure used to verify the correctness of memory chips.

In total, 28 tests were performed on each memory chip. In Table 3.8 *Test #* shows the test number. *Op. Name* presents test operations executed in the corresponding test. *Op. Info* shortly describes executed test operations, and *# of Exec.* defines how many times corresponding test operations were performed.

The test operations are described as follows:

- CW - Flash memory access mode (See Section 2.2). Common write operation provides writing of all cells in the memory matrix at once. This mode is used before every memory erase procedure.

- CER - Flash memory access mode (See Section 2.2). Common erase operation provides erasing of all cells in the memory matrix at once.

Table 3.8: Test procedure used to verify the correctness of 8k x 44-bit IHP NOR flash memory chips.

Test #	Op. Name	Op. Info	# of Exec.
1	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
2	wsolid1	Write Solid 1	1
	#rdsolid1	Read Solid 1	1
3	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
4	wcelb	Write Celb	1
	#rdcelb	Read Celb	1
5	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
6	wcelbi	Write Celbi	1
	#rdcelbi	Read Celbi	1
7	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
8	wck0	Write Check. 0	1
	#rdck0	Read Check. 0	1
9	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
10	wck1	Write Check. 1	1
	#rdck1	Read Check. 1	1
11	cw	Common Write	100
	cer	Common Erase	
	#rd0	Read 0	1
12	wck0	Write Check. 0	1
	#rdck0	Read Check. 0	1
13	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
14	wck1	Write Check. 1	1
	#rdck1	Read Check. 1	1
15	cw	Common Write	400
	cer	Common Erase	
	#rd0	Read 0	1
16	wck0	Write Check. 0	1
	#rdck0	Read Check. 0	1
17	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
18	wck1	Write Check. 1	1
	#rdck1	Read Check. 1	1
19	cw	Common Write	501
	cer	Common Erase	
	#rd0	Read 0	1
20	wsolid1	Write Solid 1	1
	#rdsolid1	Read Solid 1	1
21	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
22	wcelb	Write Celb	1
	#rdcelb	Read Celb	1
23	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
24	wcelbi	Write Celbi	1
	#rdcelbi	Read Celbi	1
25	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
26	wck0	Write Check. 0	1
	#rdck0	Read Check. 0	1
27	cw	Common Write	1
	cer	Common Erase	1
	#rd0	Read 0	1
28	wck1	Write Check. 1	1
	#rdck1	Read Check. 1	1

- RD0 - Operation used to verify the memory erase operation. That is, the whole memory matrix is read word-by-word and checked whether each bit in each memory word has the '0' state.
- WSOLID1 - Operation which writes all1 pattern to each memory word.
- RDSOLID1 - Operation used to verify the pattern stored by the WSOLID1 operation.
- WCELB - Operation used to check the memory matrix against write disturbs, and coupling and pattern-sensitive faults. During WCELB operation the middle column in each column block is written with ones except a single memory cell located in the middle of the column. In addition, a middle row in the memory is written with ones except mentioned memory cells located in the middle of the column in each column block. The resulting memory array pattern is depicted in Figure 3.5. Figure 3.5 shows the memory matrix divided into 44 column blocks outlined with thick rectangle. Black pixel in each column block defines memory cell with '1' state.

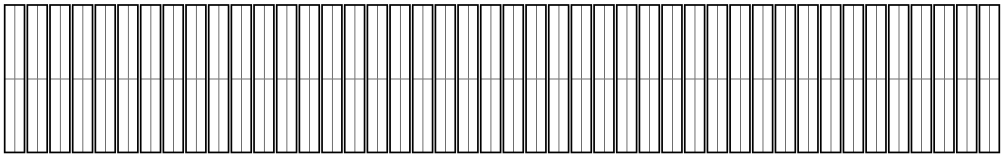


Figure 3.5: Memory array pattern generated by WCELB operation.

- RDCELB - Operation used to verify the pattern stored by the WCELB operation.
- WCELBI - Operation which generates memory array pattern inverted to one generated by the WCELB operation (Fig. 3.6).

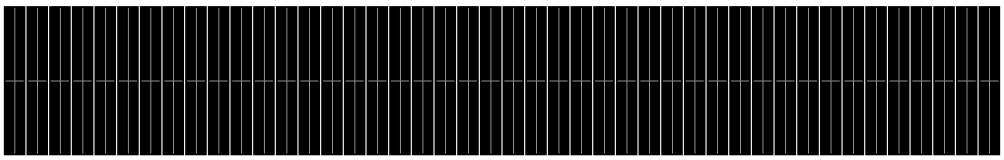


Figure 3.6: Memory array pattern generated by WCELBI operation.

- RDCELBI - Operation used to verify the pattern stored by the WCELBI operation.
- WCK0 - Operation which writes checkerboard pattern in the memory array. In the WCK0 the first cell in the first row of the memory array is written with '0', second memory cell is written with '1', the third cell with '0', and so on. In the second memory row, the first cell is written with '1', second cell with '0', third cell with '1', and so on. All memory rows are written alternately resulting in the memory pattern which resembles the layout of the chess board (Fig. 3.7).

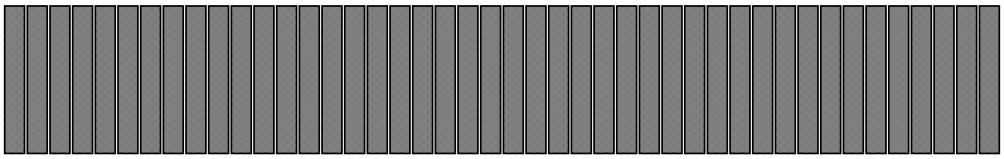


Figure 3.7: Memory array pattern generated by WCK0 operation.

- RDCK0 - Operation used to verify the pattern stored by the WCK0 operation.
- WCK1 - Operation which generates memory array pattern inverted to one generated by the WCK0 operation. WCK0 and WCK1 operations are commonly used to detect stuck-at faults in the memory array.
- RDCK1 - Operation used to verify the pattern stored by the WCK1 operation.

Each test ended with the verify operation (*RD0*, *RDSOLID1*, *RDCELB*, *RDCELBI*, *RDCK0*, or *RDCK1*). After each verify operation, the memory bitmap was stored in the file. In Table 3.8 a # sign preceding the operation name defines the operation after which the bitmap was stored. The purpose of tests 11, 15, and 19 was memory endurance screening. That is, they were used to generate wear of memory cells and to detect early failing cells.

Among 79 memory chips 38 were corrupted which translates into fabrication yield equal to ≈ 0.52. Test results concerning corrupted memory chips are presented in Table 3.9. In Table 3.9, *# of Faults* shows the number of faulty cells in the memory array. *Max. # of Faults in Word* presents maximum number of faulty cells which occurred in a single memory word. *Var. Cells* show the number of cells which changed their state from uncorrupted to corrupted multiple times during memory tests. Variable cells refer to memory cells suffering from coupling and/or pattern-sensitive faults. The *Max. # of Cell Changes* shows the maximum number of times a corrupted cell changed its state from uncorrupted to corrupted. The *Faulty Area* presents percentage of corrupted memory area. The *Comment* shows the faulty memory elements which largely contributed to the total number of memory faults. They are described as follows:

- WA - more than half or whole memory array.
- CCB - corrupted column blocks.
- CC - corrupted columns.
- CR - corrupted rows.
- CW - corrupted words.
- ABF - adjacent-bit faults.
- BF - random-bit faults.

Table 3.9: Test results concerning corrupted memory chips.

Chip ID	# of Faults	Max. # of Faults in Word	Var. Cells	Max. # of Cell Changes	Faulty Area	Comment
W02_1_5	378	2	78	3	0,105%	CC, BF
W02_3_2	360447	44	295571	12	100,000%	WA
W02_3_4	360401	44	122186	10	99,987%	WA
W02_4_7	360448	44	0	0	100,000%	WA
W02_5_6	360448	44	31204	19	100,000%	WA
W02_6_4	360448	44	33945	18	100,000%	WA
W02_6_7	2	1	0	0	0,001%	ABF
W02_8_7	276	44	215	2	0,077%	CC, CW
W02_9_6	360432	44	1765	2	99,996%	WA
W04_2_6	358031	44	2248	4	99,329%	WA
W04_3_4	231	1	8	1	0,064%	ABF
W04_3_6	148	1	1	1	0,041%	ABF
W04_4_5	360448	44	70043	22	100,000%	WA
W04_4_7	360448	44	0	0	100,000%	WA
W04_7_2	37	1	0	0	0,010%	ABF, BF
W04_7_5	2	1	2	4	0,001%	BF
W04_8_4	3	1	3	2	0,001%	BF
W04_8_5	1	1	1	6	0,000%	BF
W04_8_7	24576	3	2	4	6,818%	CCB
W04_10_4	2	1	0	0	0,001%	ABF
W05_1_3	29	1	0	0	0,008%	ABF, BF
W05_4_7	360448	44	0	0	100,000%	WA
W05_6_6	1	1	0	0	0,000%	BF
W05_7_3	128	1	0	0	0,036%	CC
W05_7_7	327576	44	314822	8	90,880%	WA
W05_8_3	446	1	411	6	0,124%	CCB
W05_8_7	16384	2	1	2	4,545%	CCB, BF
W05_9_6	15	1	10	6	0,004%	ABF, BF
W05_10_5	28789	44	17072	6	7,987%	CC
W05_11_3	360448	44	0	0	100,000%	WA
W06_2_2	2	1	0	0	0,001%	ABF
W06_3_4	2	1	0	0	0,001%	ABF
W06_4_1	640	22	556	4	0,178%	CR
W06_4_7	360448	44	0	0	100,000%	WA
W06_5_7	1	1	0	0	0,000%	BF
W06_7_6	1	1	1	6	0,000%	BF
W06_9_5	360448	44	243814	14	100,000%	WA
W06_10_6	3	1	3	7	0,001%	ABF

Data for the *Comment* column was prepared after visual inspection of bitmaps generated during memory test procedures.

The selected BWR_ERA configuration is able to replace up to 5 CCSBs (3 by BR, 1 by WR, and 1 by ERA) in each of 64 memory blocks. In addition, the BWR_ERA is able to correct a single random-bit error in each memory word. As a result, not all corrupted memory chips presented in Table 3.9 can be repaired by the proposed approach. Obviously, memory chips where more than half or whole memory array is corrupted cannot be repaired. Further, memory chips with corrupted rows or corrupted words also cannot be repaired. As a consequence, from 38 corrupted memories only 21 were used for further evaluation. The memory chips selected for generating NVM models are presented in Table 3.10.

Table 3.10: Corrupted flash memory chips selected for the evaluation.

Chip ID	# of Faults	Max. # of Faults in Word	Stuck-at-0	Stuck-at-1	Faulty Area	Comment
W02_1_5	378	2	0	402	0,105%	CC, BF
W02_6_7	2	1	0	2	0,001%	ABF
W04_3_4	231	1	0	231	0,064%	ABF
W04_3_6	148	1	0	148	0,041%	ABF
W04_7_2	37	1	0	37	0,010%	ABF, BF
W04_7_5	2	1	0	8	0,001%	BF
W04_8_4	3	1	0	5	0,001%	BF
W04_8_5	1	1	0	0	0,000%	BF
W04_8_7	24576	3	24576	7	6,818%	CCB
W04_10_4	2	1	0	2	0,001%	ABF
W05_1_3	29	1	0	29	0,008%	ABF, BF
W05_6_6	1	1	0	1	0,000%	BF
W05_7_3	128	1	0	128	0,036%	CC
W05_8_3	446	1	0	838	0,124%	CCB
W05_8_7	16384	2	16384	2	4,545%	CCB, BF
W05_9_6	15	1	6	52	0,004%	ABF, BF
W06_2_2	2	1	2	0	0,001%	ABF
W06_3_4	2	1	0	2	0,001%	ABF
W06_5_7	1	1	0	1	0,000%	BF
W06_7_6	1	1	0	6	0,000%	BF
W06_10_6	3	1	3	18	0,001%	ABF
		Total:	40971	1919		

In Table 3.10 *Stuck-at-0* and *Stuck-at-1* columns present the number of stuck-at-0 and stuck-at-1 fault occurrences (including occurrences generated by variable cells), respectively. As one can observe, the majority of selected chips have less than 1% of corrupted memory area. Moreover, in general, faults in the memory array tend to occur in clusters either as corrupted column blocks, corrupted columns, and/or adjacent bit faults (Fig. 3.8). Although the selected memory chips represent the *infant-mortality section* of the famous *bathtub curve* it may be safe to assume that:

- evaluations of memory repair techniques based on the expectation that the memory faults can occur in any place in the memory array and that new faults are not correlated with previous faults may not be precise, and
- bit-based redundancy repair mechanisms (e.g., ECP [81], SAFER [84]) may be not as effective as block-based ones (e.g., BWR_ERA).

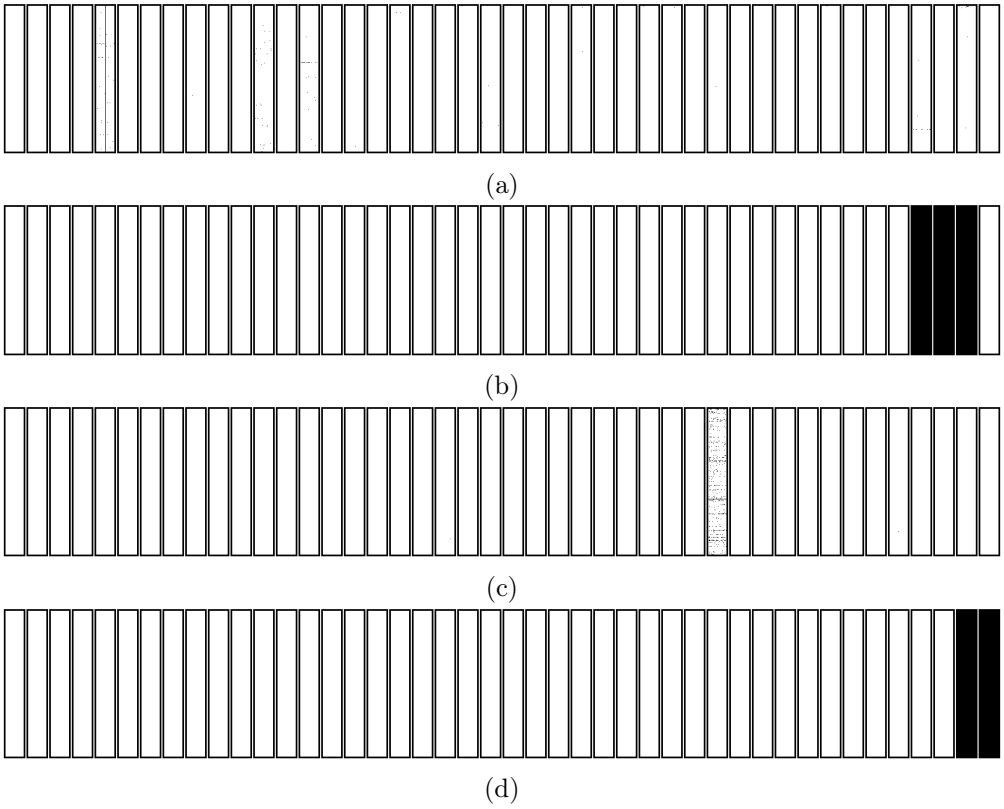


Figure 3.8: Faulty memory arrays of W02_1_5, W04_8_7, W05_8_3, and W05_8_7 memory chips.

The test results obtained for selected memory chips were used to generate memory endurance models. In order to do so, a special software for analyzing memory bitmaps and creating appropriate memory endurance models was developed. The main purpose of the software was to:

- allocate the same memory array as in the IHP 8k x 44-bit flash memory,
- allocate counter for each array element for storing the number of times the array element changed its state,
- perform the same test procedures and test operations on the allocated array as those executed during flash memory tests,

- increment the counter for each array element whenever the array element changed its state,
- check, using memory bitmaps, whenever the memory cell corresponding to the array element is corrupted, if so, disable the counter for that element.

Counters for array elements representing variable cells were disabled after the first change of variable cells from uncorrupted to corrupted. The reason for this is that the current version of the fault injection framework does not support coupling and/or pattern-sensitive faults. Moreover, because some selected chips had only few memory array faults (e.g., only one in *W06_7_6* chip) their utilization for the generation of NVM models would not be valuable. Because of that, for uncorrupted array elements a random endurance values were assigned according to the Normal distribution with mean equal to 4000 and standard deviation equal to 400. Low values for mean and standard deviation were chosen deliberately to speed-up simulation and emulation procedures.

As a consequence, memory endurance models generated by the analysis software represented two fault distributions, one based on real memory devices and second based on the Normal distribution. The analysis software was executed for each selected chip. As a result, 21 different memory endurance models were generated.

Memory usage model

The last step in creating appropriate NVM models for the purpose of the BWR_ERA evaluation was to generate memory usage model. As described in the previous section, the memory usage model is responsible for providing appropriate accesses to the memory. That is, it generates appropriate memory access requests, memory addresses, and memory input data. Further, for proper generation of NVM models, the memory usage model should also include any fault-tolerance and life-prolonging techniques that will be implemented in the system. The reason for this is that fault-tolerance and life-prolonging techniques change original memory accesses, addresses, and input data. As a consequence, the memory usage model used for the evaluation consisted of two modules: memory usage stimuli, and BWR_ERA system (Fig. 3.9).

Memory usage stimuli module - The memory usage stimuli generates original accesses to the memory. That is, it represents the way how the the memory is used by the user/system. For example, for the evaluation of the fault injection framework, uniform and pareto memory usage models were used. For the evaluation of the BWR_ERA system, the memory usage stimuli consisted of two models.

The first model generated the same memory accesses as used during memory test procedures. The reason for that was to provide the same memory operations and input data which would activate all memory array faults that occurred during memory test procedures.

The second model represented mentioned uniform model. That is, in the uniform model random data is stored in the memory and read back. Next, memory address

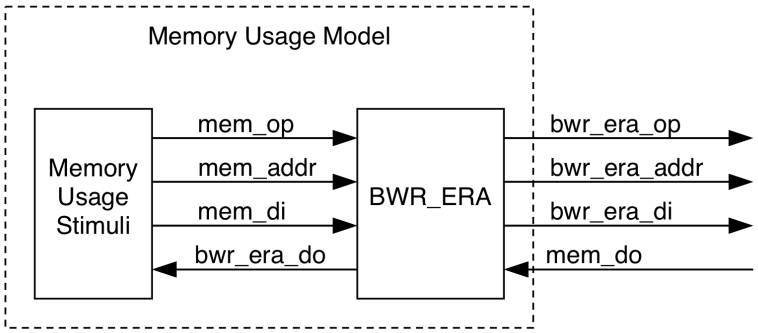


Figure 3.9: Memory usage model used for evaluation.

is incremented, and another random data is stored and read back. The procedure is continued until all words in the memory are filled with random data. Next, the memory erase operation is executed, and memory address is set to 0. Further, the write-read-back procedure is executed, and so on. The uniform model was used to wear-out memory cells and to produce memory faults.

BWR_ERA module - The BWR_ERA module implemented in the memory usage model represented the golden model of the BWR_ERA system. That is, it represented the BWR_ERA mechanism as it would be implemented in a real system.

BWR_ERA simulation

Finally, having memory endurance models and memory usage model, the generation of NVM models and simulation of the BWR_ERA could be performed. The main reason why NVM models were used for evaluation purposes instead of real devices is the better usability of NVM models. That is, in contrast to real flash memory chips, NVM models can be used in many evaluations. Memory cells in flash memories, once corrupted, cannot become uncorrupted. As a consequence, the flash memory chips can be generally used only in one evaluation. Furthermore, because NVM models employ SRAM as data storage, the simulations and emulations performed with NVM models are simply faster.

The configuration data required to produce appropriate behavior for NVM models was generated with the report system. Because the BWR_ERA module was implemented in the memory usage model, the memory usage simulations performed with the report system also simulated the BWR_ERA system. The simulation framework is depicted in Figure 3.10.

The simulation routine was proceeded in the following way:

1. **Memory usage simulations performed by the report system using the first model from the memory usage stimuli module.** The purpose of the first part of the simulation was to activate faults in the memory array which occurred during flash memory test procedures. Further, activated faults were managed by

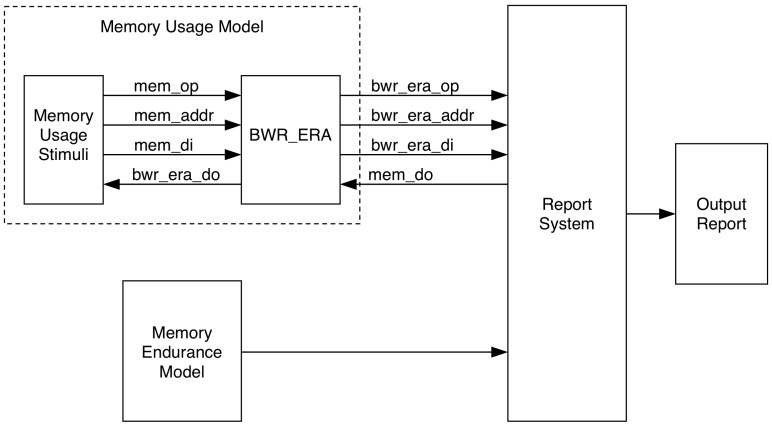


Figure 3.10: System used for simulations.

the BWR_ERA. Moreover, during first part of the simulation no block-read and block-restore operations were performed before and after the block-level repair. The reason for this is that during the first part of the simulation the content of the memory was not relevant. The main aim was to manage memory array faults only.

2. **Memory usage simulations performed by the report system using the second model from the memory usage stimuli module.** The purpose of the second part of the simulation was to evaluate if the BWR_ERA can handle additional memory array faults than those which occurred in the first part of the simulation. Moreover, in the second part of the simulation, block-read and block-restore operations were performed before and after the block-level repair. Furthermore, whenever the random data was stored in the memory and read back, the output data from the memory was compared with the input data in order to detect any discrepancies. The second part of the simulation stopped when any discrepancies between the memory input and output data were found or when the BWR_ERA reported that it cannot handle any more faults.

The memory usage simulations were performed for all memory endurance models. All simulations were ended by the BWR_ERA which defines proper implementation of the BWR_ERA mechanism.[9] After each simulation performed by the report system the output report was generated. As described in the previous section, the output report stores configuration data for the NVM model which defines the behavior of the NVM model based on the memory endurance model and the memory usage model.

The results from the simulations performed by the report system are presented in Table 3.11. In Table 3.11 *Model ID* defines NVM model generated for the corresponding memory chip. The *Real Faults* shows the number of faults reported after test procedures performed for selected memory chips. The *Sim. Faults* shows the number of faults

[9]In memory usage simulations only hard faults were occurring in the modeled memory. In the proper implementation of the BWR_ERA all of them had to be detected.

managed by the BWR_ERA system. The *Difference* shows the difference between the number of *Real Faults* and *Sim. Faults*. In other words, the *Difference* shows the number of random-bit faults modeled with the Normal distribution and corrected by the BWR_ERA.

As one can observe, the BWR_ERA provides great memory reliability improvement, especially for memories which suffer from clustered faults. For example, the BWR_ERA implemented for *W04_8_7* and *W05_8_7* NVM models is able to handle 24939 and 16773 memory array faults, respectively. As one recall from Figures 3.8b and 3.8d, *W04_8_7* and *W05_8_7* memory chips suffer from CCBs. Furthermore, not only all real faults in modeled memories were repaired, the BWR_ERA could also handle additional faults occurred in the memory array. On average the implemented BWR_ERA was able to repair $\sim 559,95$ (*Average* in Table 3.11) faulty cells modeled with the Normal distribution. However, in *W04_8_7* and *W05_8_7* memory models the repair capabilities of the BWR_ERA were limited due to CCBs in the spare area. As a consequence the BWR_ERA implemented for *W04_8_7* and *W05_8_7* memory models was able to manage 363 and 389 faulty cells modeled with the Normal distribution, respectively. If those results were excluded from the calculation of the average number of random-bit faults then on average the BWR_ERA was able to repair ~ 579 faults (*Average2* in Table 3.11) modeled with the Normal distribution. As one recall results from preliminary simulations performed for BWR_ERA presented in Table 2.15 in Section 2.5, one can notice that for memory yield equal to 0.5 the simulated BWR_ERA system was able to repair 584 random-bit faults on average. Therefore, the main conclusion which can be drawn here is that the yield simulations presented in Section 2.5 were properly performed. Furthermore, results from mentioned yield simulations present the minimal number of faults which can be corrected by the BWR_ERA. When the faults in the memory array tend to cluster, the memory reliability improvement achieved with the BWR_ERA is even greater.

Another interesting results derived from simulations concern the overhead in terms of the number of additional memory operations imposed by the BWR_ERA mechanism. The mentioned results are presented in Table 3.12. In Table 3.12 columns R, W, E, and B show the number of memory read, write, erase, and block-erase operations generated by memory usage stimuli model and the BWR_ERA module, respectively. As one can observe, the biggest overhead imposed by the BWR_ERA are memory read and block-erase operations. The requirement for larger number of read operations was caused mainly by the write-verify scheme, where after every memory write operation a memory read operation was performed. In addition, the BWR_ERA imposes large number of block-erase operations because they are required by the BR procedure. If the BWR_ERA would have been implemented for NVMs which does not suffer from erase-before-write characteristic like flash memories, then no block-erase operation would be required. In addition, one has to note that presented results concern overheads imposed by the repair mechanisms implemented in the BWR_ERA only. That is, numbers of memory read and write operations generated by the memory usage stimuli model include memory operations required for block-read and block-restore procedures.

Table 3.11: BWR_ERA simulations performed by the report system.

Model ID	Real Faults	Sim. Faults	Difference
W02_1_5	378	909	531
W02_6_7	2	583	581
W04_3_4	231	810	579
W04_3_6	148	727	579
W04_7_2	37	612	575
W04_7_5	2	583	581
W04_8_4	3	584	581
W04_8_5	1	582	581
W04_8_7	24576	24939	363
W04_10_4	2	583	581
W05_1_3	29	609	580
W05_6_6	1	583	582
W05_7_3	128	608	480
W05_8_3	446	994	548
W05_8_7	16384	16773	389
W05_9_6	15	655	640
W06_2_2	2	627	625
W06_3_4	2	583	581
W06_5_7	1	583	582
W06_7_6	1	582	581
W06_10_6	3	642	639
		Average:	559,952
		Average2:	579,316

3.3 System emulation

The purpose of the second part of evaluation was to verify the applicability of the BWR_ERA for an embedded system and to evaluate the real impact of the BWR_ERA mechanism on system's performance. Therefore, for the purpose of the evaluation, systems consisting of BWR_ERA, Leon2 processor, and generated NVM models were implemented in the GR-CPCI-XC2V Virtex2 FPGA board [73] and emulated. As mentioned in the beginning of this chapter, in order to minimize the impact on system's performance the BWR_ERA was implemented in the form of the memory controller. The single system used for the emulation is depicted in Figure 3.11 and consists of:

- HDL model of the Leon2 processor,
- HDL model of the BWR_ERA system implemented in the form of the memory controller and connected to the Leon2 processor,
- NVM model connected to the BWR_ERA controller, and
- the test program for the Leon2 processor.

Table 3.12: Additional memory operations imposed by memory usage stimuli model and BWR_ERA module.

Model ID	Memory Usage Stimuli				BWR_ERA Module			
	R	W	E	B	R	W	E	B
W02_1_5	3435369	3288004	1401	0	6771643	3312254	1401	189
W02_6_7	3487701	3340594	1406	0	6876815	3364974	1406	190
W04_3_4	3487061	3339942	1406	0	6875523	3364321	1406	190
W04_3_6	3487061	3339988	1406	0	6875317	3364240	1406	189
W04_7_2	3479514	3332470	1405	0	6860756	3356978	1405	191
W04_7_5	3487701	3340557	1406	0	6876779	3364937	1406	190
W04_8_4	3487573	3340472	1406	0	6876820	3364980	1406	191
W04_8_5	3487573	3340458	1406	0	6876552	3364838	1406	190
W04_8_7	3264813	3117692	1380	0	6428215	3140663	1380	179
W04_10_4	3487701	3340594	1406	0	6876815	3364974	1406	190
W05_1_3	3487318	3340046	1406	0	6875625	3364297	1406	189
W05_6_6	3487830	3340650	1406	0	6876998	3365030	1406	190
W05_7_3	3386222	3238955	1395	0	6673694	3263336	1395	190
W05_8_3	3456944	3309620	1403	0	6815335	3334126	1403	191
W05_8_7	3296749	3149560	1383	0	6491498	3172266	1383	177
W05_9_6	3544987	3397874	1413	0	6991382	3422252	1413	190
W06_2_2	3528791	3381610	1411	0	6958923	3405990	1411	190
W06_3_4	3487829	3340648	1406	0	6876997	3365028	1406	190
W06_5_7	3487573	3340509	1406	0	6876602	3364889	1406	190
W06_7_6	3487573	3340458	1406	0	6876552	3364838	1406	190
W06_10_6	3545115	3398003	1413	0	6991639	3422383	1413	190

In order to store configuration data (EPP, CSP, and ERA_POS vectors) required by the BWR_ERA to perform repair procedures, small SRAM memory was implemented in the BWR_ERA controller.

The test program used in the system was responsible for configuring the NVM model using configuration data from the output report. Moreover, the test program was used to perform memory usage emulations using modified memory usage model presented in the previous section. Because the BWR_ERA was implemented in the system, there was no need to include it in the memory usage model. As a result, the memory usage model consisted of the memory usage stimuli module only. Finally, on each step of the memory usage emulation, the test program screened fault injection module registers to check if the faults in the memory array were injected correctly. Moreover, it also screened BWR_ERA controller registers to check if the BWR_ERA is still able to correct memory faults.

Because the main purpose of this part of the evaluation was to check the applicability of the BWR_ERA for the embedded system not all NVM models generated in the previous section were used for system emulations. Unfortunately, NVM models with the largest amount of memory array faults (*W04_8_7* and *W05_8_7* NVM models) could not be used in system emulations. The amount of storage required for the fault injection module

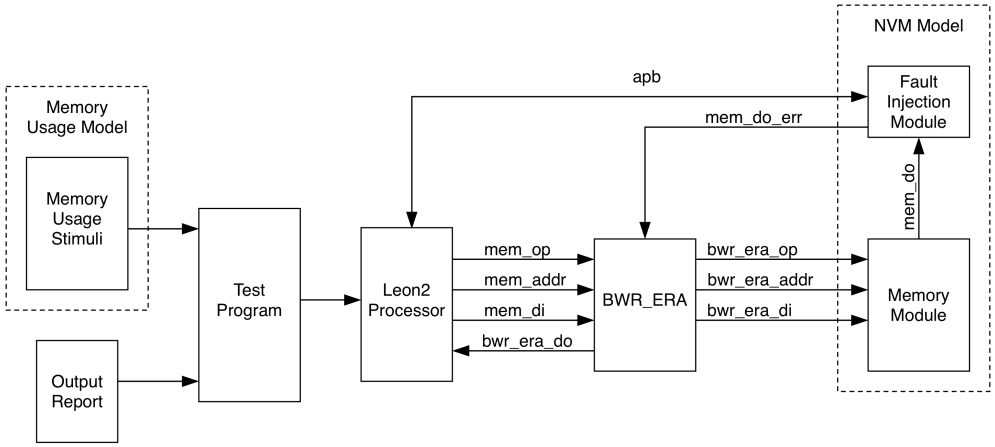


Figure 3.11: System used for emulations.

for those models exceeded the amount of SRAM resources in the FPGA. Because of that, 4 NVM models with the largest number of memory array faults (*W02_1_5*, *W04_3_4*, *W04_3_6*, and *W05_8_3*), except *W04_8_7* and *W05_8_7* models, were selected for system emulations.

The evaluation was performed in the following way:

1. A single NVM model from selected models was connected to the BWR_ERA controller.
2. The resulting system was synthesized and implemented in the FPGA.
3. The test program including appropriate configuration data for selected NVM model and memory usage model was compiled and uploaded into the Leon2 processor.
4. The test program was executed on the processor and the 64-bit on-chip timer was enabled.
5. Memory usage emulations were stopped whenever faults in the memory array were incorrectly injected, or the BWR_ERA was not able to repair more faults.
6. After memory usage emulations, the 64-bit on-chip timer was read and its value was reported.
7. Another NVM from selected models was connected to the BWR_ERA controller and another system emulation was performed.

Furthermore, similarly to the evaluation of the fault injection framework presented in Section 3.1, for each developed system a reference system was also evaluated. In the reference system no fault injection and no fault repair were performed. Moreover, the BWR_ERA controller did not execute the verify operation after each memory write

operation. The BWR_ERA controller in the reference system behaved as a standard memory controller. Finally, because no repair was performed in the reference system, the test program did not execute block-read and block-restore operations.

In order to properly evaluate the impact of the BWR_ERA on system's performance, memory access times used for NVM models had to correspond to IHP 8k x 44-bit NOR flash memory access times. However, the 12 MHz clock used in the system produced period longer than the minimal access time (flash read access time) required by the flash memory. As a consequence, flash memory access times were normalized to flash read time. Further, normalized values were used as numbers of clock cycles required to perform appropriate memory accesses. Clock cycles required for NVM model accesses are presented in Table 3.13.

Table 3.13: Flash memory access times and clock cycles used for NVM model accesses.

Mode	Unit	Time	Normalized Values
CWr	μs	500	10000
CEr	ms	50	1000000
SWr	μs	50	1000
SEr	ms	50	1000000
PreWrite	μs	300	6000
Write	μs	100	2000
PreRead	ns	100	2
Read	ns	50	1

All system emulations were performed on the GR-CPCI-XC2V Virtex2 FPGA board [73] using 12 MHz system clock. During all system emulations all faults in memory arrays were injected correctly. Further, no discrepancies between memory input and output data were reported. In addition, the number of errors corrected by the BWR_ERA was the same as reported after BWR_ERA simulations performed with the report system.

Results from system emulations are presented in Table 3.14. In Table 3.14 *Model ID* defines appropriate NVM model. *Setup Time* shows the time required to configure the NVM model. That is, it is the time required to store appropriate data in the memories inside the fault injection module. *Memory Usage Time* is the time of memory usage emulations. *BARE* defines reference system and *BWR_ERA* defines system where BWR_ERA performed repair procedures. Finally, *Ratio* shows the ratio of the memory usage emulation time in the system where BWR_ERA performed repair to the memory usage emulation time in the reference system.

As one can observe, the overhead in terms of additional memory accesses imposed by repair procedures is not so big. In general, BWR_ERA caused less than 5% of system's performance degradation. The performance degradation was mainly caused by block-erase and block-restore operations. The reason for this is that they require a long time to be performed. One has to remember that the block-erase operations are only required for flash memories. If the BWR_ERA would have been implemented for memories which do not suffer from erase-before-write characteristic, then the imposed degradation would be lower. Finally, although the number of memory read operations is almost doubled in

Table 3.14: Results from system emulations.

	Setup Time [s]	Memory Usage Time [s]		
Model ID	BWR_ERA	BARE	BWR_ERA	Ratio
W02_1_5	0,0531	2446,5820	2554,5123	1,0441
W04_3_4	0,0534	2477,9090	2592,5938	1,0463
W04_3_6	0,0472	2477,4455	2592,9116	1,0466
W05_8_3	0,0525	2457,5194	2569,4360	1,0455

systems where BWR_ERA is implemented (see Tab. 3.12 in Section 3.2), it has not so significant impact on system's performance. At least it is not so significant in emulated systems.

Chapter 4

Conclusion and future work

Technological progress that occurred in recent years has had a huge impact on our lives. The unprecedented development of mobile technologies provided us with the access to an enormous amount of information. As a consequence, the development of mobile devices has skyrocketed in order to provide us tools for consuming this information.

As more and more information is being stored and processed, faster, denser, and more power-efficient semiconductor memories are required. In order to satisfy this need, new memory technologies are being developed. Among them, non-volatile memory technologies arouse the greatest interest as they are expected to fulfill requirements of upcoming mobile devices. However, before they can be implemented in digital systems, concerns related to their reliability have to be addressed.

The main driving force of the research conducted for the purpose of the PhD project was to address these reliability concerns. This was achieved by developing novel memory repair techniques that could enable wider adoption of NVMs in digital systems. Each of developed techniques presented in previous chapters focus on different reliability aspects of NVMs, i.e.:

- The word-level repair provides on-line repair of wear-out memory cells that can occur during memory lifetime.
- The block-level repair focuses on post-production faults in the memory array.
- The error-correcting code with increased hard error correction capability provides protection against hard faults that can occur in the memory array and soft errors caused e.g., by external factors.

All proposed techniques were evaluated, and their repair capabilities confirmed. Moreover, it was shown that by combining the proposed techniques into a consistent system the synergistic effect can be produced which provides even greater memory reliability improvement. Further, the origin of the synergistic effect was explained and the achieved memory reliability improvement was quantified. In addition, the applicability of developed mechanisms for embedded systems was proven through system emulations.

Although most of research objectives were met, some aspects of the proposed techniques require further research. They are briefly described in further sections.

4.1 Configuration data management

Techniques proposed in the thesis use additional memory area in order to store configuration data required to perform repair operations. For the purpose of evaluations it was assumed that the configuration data is stored in a fast and reliable memory. In system emulations described in the previous chapter the configuration data was stored in a small SRAM implemented in the BWR_ERA controller.

The necessity for a fast memory for configuration data is predicted by the fact that it has to be read on every access to the NVM. Therefore, when the access speed is of utmost importance, the best candidates for the storage medium are SRAM and register file. However, SRAM and register file are volatile memories which means that the configuration data would be lost in case when the system is powered off. In order to solve this problem, the configuration data would have to be stored in the non-volatile memory e.g., when the system is being powered off. Further, after system power up, the configuration data would have to be read from the non-volatile memory and stored in the SRAM or register file.

Introducing another non-volatile memory in the system would complicate the system design process and it would have a negative impact on the reliability of the system in general. As a consequence, the configuration data should be stored in the same non-volatile memory for which the proposed techniques were implemented (e.g., at the and of the memory area). This, however, raises another concerns related to the reliability of the area in the NVM reserved for the configuration data. Possible solutions to this problem would be to protect the configuration data with the ECC or with the N-modular redundancy, or both.

Another aspect related to the correctness of the configuration data is related to the reliability of the SRAM or register file. Because SRAM and register file are prone to soft errors, it may happen that during system operation data stored in them will be corrupted. Therefore, they also should be protected with ECC or N-modular redundancy.

The reliability of the configuration data is very important for the reliability of the whole system. As a consequence, further investigations should be conducted in order to provide solutions which will preserve the correctness of the configuration data.

4.2 Modularity and flexibility of the comprehensive approach

In the comprehensive approach presented in the thesis first faults in the memory are managed by the BR mechanism. Further faults in the memory array are handled by the WR. When all SPs belonging to the BR and WR are used, following memory faults are corrected by the ERA technique. The combination of the BR, WR, and ERA mechanisms in the comprehensive approach is best suited for memories which suffer from large number of post-production faults. The reason for this is that the post-production faults can be managed by the BR right after manufacturing while following faults occurring during memory lifetime can be managed by fast (in comparison to BR) WR and ERA techniques. As a consequence, system's performance would not be degraded by the BR procedure.

However, because presented repair techniques can be combined in many ways, different repair schemes can be achieved. For example, one can combine BR, WR, and ERA in a way in which first faults in the memory array are handled by the WR and ERA while the BR mechanism is used only to restore the repair capabilities of the WR and ERA. That is, whenever fault occur in the memory array it is first corrected by the WR or ERA. Further, when the memory is not used, the BR can be executed to replace CCSB previously replaced by the WR or ERA. Next, appropriate WR EPP or ERA_POS which were used to store the position of the CCSB can be marked as unused.

Another possible combination concerns a system where the BR is used for post-production faults while ERA manages following faults in the memory array. Next, whenever the double-bit fault is detected in the memory word, WR EPP is used instead of ERA_POS for managing the detected erasure. In this way, the birthday paradox can be circumvented by the WR. Furthermore, it can be circumvented until all SPs belonging to the WR are used. As a consequence, probably even greater synergistic effect can be achieved than that produced in a comprehensive approach presented in the thesis.

To conclude, the repair schemes and capabilities achieved with different combinations of presented techniques require further investigations.

Bibliography

[1] Advantest. *V93000 SOC*. URL: `http://www.verigy.com/ate/products/V93000/index.htm`.

[2] P.P. Ankolekar, R. Isaac, and J.W. Bredow. "Multibit Error-Correction Methods for Latency-Constrained Flash Memory Systems". In: *Device and Materials Reliability, IEEE Transactions on* 10.1 (Mar. 2010), pp. 33–39.

[3] W.D. Armitage and J.-C. Lo. "Erasure error correction with hardware detection". In: *Defect and Fault Tolerance in VLSI Systems, 1999. DFT '99. International Symposium on*. Nov. 1999, pp. 293–301.

[4] M.N. Ashraf and J. Dastur. "Software Based NAND Flash Management Techniques". In: *Computing, Engineering and Information, 2009. ICC '09. International Conference on*. Apr. 2009, pp. 168–171.

[5] C. Augustine et al. "STT-MRAMs for future universal memories: Perspective and prospective". In: *Microelectronics (MIEL), 2012 28th International Conference on*. May 2012, pp. 349–355.

[6] M. Bagatin et al. "Single Event Effects in 1Gbit 90nm NAND Flash Memories under Operating Conditions". In: *On-Line Testing Symposium, 2007. IOLTS 07. 13th IEEE International*. July 2007, pp. 146–151.

[7] F. Bedeschi et al. "A Multi-Level-Cell Bipolar-Selected Phase-Change Memory". In: *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*. Feb. 2008, pp. 428–625.

[8] Benso Alfredo et al. "Specification and design of a new memory fault simulator". In: *Proceedings of the 11th Asian Test Symposium, 2002. (ATS '02)*. IEEE Comput. Soc, 2002, pp. 92–97.

[9] A. Bette et al. "A high-speed 128 Kbit MRAM core for future universal memory applications". In: *VLSI Circuits, 2003. Digest of Technical Papers. 2003 Symposium on*. June 2003, pp. 217–220.

[10] R. Buhrman. "Spin torque MRAM - Challenges and prospects". In: *Device Research Conference, 2009. DRC 2009*. June 2009, pp. 33–33.

[11] B.H. Calhoun et al. "Digital Circuit Design Challenges and Opportunities in the Era of Nanoscale CMOS". In: *Proceedings of the IEEE* 96.2 (Feb. 2008), pp. 343–365.

[12] G. Campardo, R. Micheloni, and D. Novosel. *VLSI-Design of Non-Volatile Memories*. Springer, 2005.

[13] M. Caramia et al. “Automated synthesis of EDACs for FLASH memories with user-selectable correction capability”. In: *High Level Design Validation and Test Workshop (HLDVT), 2010 IEEE International*. June 2010, pp. 113–120.

[14] L.V. Cargnini et al. “Improving the Reliability of a FPGA Using Fault-Tolerance Mechanism Based on Magnetic Memory (MRAM)”. In: *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*. Dec. 2010, pp. 150–155.

[15] Bainan Chen, Xinmiao Zhang, and Zhongfeng Wang. “Error correction for multi-level NAND flash memory using Reed-Solomon codes”. In: *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*. Oct. 2008, pp. 94–99.

[16] C.L. Chen and M.Y. Hsiao. “Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review”. In: *IBM Journal of Research and Development* 28.2 (Mar. 1984), pp. 124–134.

[17] Te-Hsuan Chen et al. “An Adaptive-Rate Error Correction Scheme for NAND Flash Memory”. In: *VLSI Test Symposium, 2009. VTS ’09. 27th IEEE*. May 2009, pp. 53–58.

[18] W.K. Chen. *Memory, Microprocessor, and ASIC*. Principles and Applications in Engineering. Taylor & Francis, 2004.

[19] Kuo-Liang Cheng et al. “RAMSES-FT: a fault simulator for flash memory testing and diagnostics”. In: *Proceedings 20th IEEE VLSI Test Symposium (VTS 2002)*. IEEE Comput. Soc, 2002, pp. 281–286.

[20] Sangyeun Cho and Hyunjin Lee. “Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance”. In: *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. Dec. 2009, pp. 347–357.

[21] Hyojin Choi, Wei Liu, and Wonyong Sung. “VLSI Implementation of BCH Error Correction for Multilevel Cell NAND Flash Memory”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 18.5 (May 2010), pp. 843–847.

[22] Ki Chul Chun et al. “A Scaling Roadmap and Performance Evaluation of In-Plane and Perpendicular MTJ Based STT-MRAMs for High-Density Cache Memory”. In: *Solid-State Circuits, IEEE Journal of* 48.2 (Feb. 2013), pp. 598–610.

[23] B.F. Cockburn. “Tutorial on magnetic tunnel junction magnetoresistive random-access memory”. In: *Memory Technology, Design and Testing, 2004. Records of the 2004 International Workshop on*. Aug. 2004, pp. 46–51.

[24] Semico Research Corporation. *Semico: System(s)-on-a-Chip - A Braver New World | Semico Research*. URL: `http://www.semico.com/content/semico-systems-chip-âĂŞ-braver-new-world` (visited on 05/01/2014).

[25] Cypress. *Nonvolatile Products - Cypress*. URL: `http://www.cypress.com/nonvolatile/?source=CY-ENG-HEADER` (visited on 05/21/2015).

[26] S. Eilert, M. Leinwander, and G. Crisenza. "Phase Change Memory: A New Memory Enables New Memory Usage Models". In: *Memory Workshop, 2009. IMW '09. IEEE International*. May 2009, pp. 1–2.

[27] A.P. Ferreira et al. "Using PCM in Next-generation Embedded Space Applications". In: *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*. Apr. 2010, pp. 153–162.

[28] R. F. Freitas and W.W. Wilcke. "Storage-class memory: The next storage system technology". In: *IBM Journal of Research and Development* 52.4.5 (July 2008), pp. 439–447.

[29] Yoshihisa Fujisaki. "Current Status of Nonvolatile Semiconductor Memory Technology". In: *Japanese Journal of Applied Physics* 49.10R (2010).

[30] Fujitsu. *Ferroelectric RAM (FeRAM) : Fujitsu Europe, Middle East and Africa*. URL: `http://www.fujitsu.com/emea/services/microelectronics/fram/` (visited on 05/21/2015).

[31] E. Gal and Sivan Toledo. "Mapping structures for flash memories: techniques and open problems". In: *Software - Science, Technology and Engineering, 2005. Proceedings. IEEE International Conference on*. Feb. 2005, pp. 83–92.

[32] S. Gerardin and A. Paccagnella. "Present and Future Non-Volatile Memories for Space". In: *Nuclear Science, IEEE Transactions on* 57.6 (Dec. 2010), pp. 3016–3039.

[33] O. Ginez, J. Portal, and H. Aziza. "An on-line testing scheme for repairing purposes in Flash memories". In: *Design and Diagnostics of Electronic Circuits Systems, 2009. DDECS '09. 12th International Symposium on*. Apr. 2009, pp. 120–123.

[34] O. Ginez, J. M Portal, and H. Aziza. "Reliability issues in flash memories: An online diagnosis and repair scheme for word line drivers". In: *Mixed-Signals, Sensors, and Systems Test Workshop, 2008. IMS3TW 2008. IEEE 14th International*. June 2008, pp. 1–6.

[35] B. Gleixner, F. Pellizzer, and R. Bez. "Reliability characterization of Phase Change Memory". In: *Non-Volatile Memory Technology Symposium (NVMTS), 2009 10th Annual*. Oct. 2009, pp. 7–11.

[36] B. Godard et al. "Architecture for Highly Reliable Embedded Flash Memories". In: *Design and Diagnostics of Electronic Circuits and Systems, 2007. DDECS '07. IEEE*. Apr. 2007, pp. 1–6.

[37] B. Godard et al. "Hierarchical Code Correction and Reliability Management in Embedded nor Flash Memories". In: *Test Symposium, 2008 13th European*. May 2008, pp. 84–90.

[38] L.M. Grupp et al. "Characterizing flash memory: Anomalies, observations, and applications". In: *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. Dec. 2009, pp. 24–33.

[39] T.P. Haraszti. *CMOS Memory Circuits*. Springer, 2000.

[40] M. Horiguchi and K. Itoh. *Nanoscale Memory Repair*. Integrated Circuits and Systems. Springer, 2011.

[41] M.Y. Hsiao. "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes". In: *IBM Journal of Research and Development* 14.4 (July 1970), pp. 395–401.

[42] Yu-Ying Hsiao, Chao-Hsun Chen, and Cheng-Wen Wu. "A built-in self-repair scheme for NOR-type flash memory". In: *VLSI Test Symposium, 2006. Proceedings. 24th IEEE*. Apr. 2006, 6 pp.–119.

[43] Yu-Ying Hsiao, Chao-Hsun Chen, and Cheng-Wen Wu. "Built-In Self-Repair Schemes for Flash Memories". In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 29.8 (Aug. 2010), pp. 1243–1256.

[44] Rei-Fu Huang, Chao-Hsun Chen, and Cheng-Wen Wu. "Economic Aspects of Memory Built-in Self-Repair". In: *Design Test of Computers, IEEE* 24.2 (Mar. 2007), pp. 164–172.

[45] Texas Instruments. *MSP430 FRAM Series - Overview - TI.com*. URL: `http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/fram/overview.page` (visited on 05/21/2015).

[46] Engin Ipek et al. "Dynamically Replicated Memory: Building Reliable Systems from Nanoscale Resistive Memories". In: *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XV. New York, NY, USA: ACM, 2010, pp. 3–14.

[47] F. Irom and D.N. Nguyen. "Single Event Effect Characterization of High Density Commercial NAND and NOR Nonvolatile Flash Memories". In: *Nuclear Science, IEEE Transactions on* 54.6 (Dec. 2007), pp. 2547–2553.

[48] B. Jacob, S. Ng, and D. Wang. *Memory Systems: Cache, DRAM, Disk*. Elsevier Science, 2010.

[49] Lei Jiang et al. "Constructing large and fast multi-level cell STT-MRAM based cache for embedded processors". In: *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. June 2012, pp. 907–912.

[50] H. Kaneko. "Error control coding for semiconductor memory systems in the space radiation environment". In: *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on*. Oct. 2005, pp. 93–101.

[51] M. Karunaratne and B. Oomann. "Yield gain with memory BISR - a case study". In: *Circuits and Systems, 2009. MWSCAS '09. 52nd IEEE International Midwest Symposium on*. Aug. 2009, pp. 699–702.

[52] S. Khan and S. Hamdioui. "Trends and challenges of SRAM reliability in the nano-scale era". In: *Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2010 5th International Conference on*. Mar. 2010, pp. 1–6.

[53] Changgeun Kim et al. "Product Reed-Solomon Codes for Implementing NAND Flash Controller on FPGA Chip". In: *Computer Engineering and Applications (ICCEA), 2010 Second International Conference on*. Vol. 1. Mar. 2010, pp. 281–285.

[54] Jingfei Kong and Huiyang Zhou. "Improving privacy and lifetime of PCM-based main memory". In: *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*. June 2010, pp. 333–342.

[55] B.C. Lee et al. "Phase-Change Technology and the Future of Main Memory". In: *Micro, IEEE* 30.1 (Jan. 2010), pp. 143–143.

[56] Hai Li and Yiran Chen. *Nonvolatile Memory Design: Magnetic, Resistive, and Phase Change*. CRC Press, Taylor & Francis Group, 2012.

[57] Jing Li, Binquan Luan, and Chung Lam. "Resistance drift in phase change memory". In: *Reliability Physics Symposium (IRPS), 2012 IEEE International*. Apr. 2012, pp. 6C.1.1–6C.1.6.

[58] Bo Liu, J.F. Frenzel, and R.B. Wells. "A multi-level DRAM with fast read and low power consumption". In: *Microelectronics and Electron Devices, 2005. WMED '05. 2005 IEEE Workshop on*. Apr. 2005, pp. 59–62.

[59] Jamie Liu et al. "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms". In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ISCA '13. Tel-Aviv, Israel: ACM, 2013, pp. 60–71.

[60] Xiaohua Lou et al. "Demonstration of multilevel cell spin transfer switching in MgO magnetic tunnel junctions". In: *Applied Physics Letters* 93.24 (Dec. 2008), pp. 242502–242502–3.

[61] Shyue-Kung Lu et al. "Synergistic Reliability and Yield Enhancement Techniques for Embedded SRAMs". In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 32.1 (Jan. 2013), pp. 165–169.

[62] T.C. MacLeod et al. "Satellite test of radiation impact on Ramtron 512K FRAM". In: *Non-Volatile Memory Technology Symposium (NVMTS), 2009 10th Annual*. Oct. 2009, pp. 24–27.

[63] Alexander Makarov, Viktor Sverdlov, and Siegfried Selberherr. "Emerging memory technologies: Trends, challenges, and modeling methods." In: *Microelectronics Reliability* 52.4 (2012), pp. 628–634.

[64] E.J. Marinissen et al. "Challenges in embedded memory design and test". In: *Design, Automation and Test in Europe, 2005. Proceedings*. Mar. 2005, 722–727 Vol. 2.

[65] George Marsaglia. *Random numbers for C: The END?* [Accessed 6-May-2013]. URL: `http://groups.google.co.uk/group/sci.math.num-analysis/msg/eb4ddde782b17051`.

[66] R. Micheloni et al. "Non-Volatile Memories for Removable Media". In: *Proceedings of the IEEE* 97.1 (Jan. 2009), pp. 148–160.

[67] N. Mielke et al. "Bit error rate in NAND Flash memories". In: *Reliability Physics Symposium, 2008. IRPS 2008. IEEE International*. Apr. 2008, pp. 9–19.

[68] Vidyabhushan Mohan et al. "How I Learned to Stop Worrying and Love Flash Endurance". In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Storage and File Systems*. HotStorage'10. Boston, MA: USENIX Association, 2010, pp. 3–3.

[69] T. Murotani et al. "A 4-level storage 4 Gb DRAM". In: *Solid-State Circuits Conference, 1997. Digest of Technical Papers. 43rd ISSCC., 1997 IEEE International*. Feb. 1997, pp. 74–75.

[70] Sang Phill Park et al. "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture". In: *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. June 2012, pp. 492–497.

[71] K. Pekmestzi et al. "A BISR Architecture for Embedded Memories". In: *On-Line Testing Symposium, 2008. IOLTS '08. 14th IEEE International*. July 2008, pp. 149–154.

[72] F. Pellizzer et al. "A 90nm Phase Change Memory Technology for Stand-Alone Non-Volatile Memory Applications". In: *VLSI Technology, 2006. Digest of Technical Papers. 2006 Symposium on*. 2006, pp. 122–123.

[73] PENDER Electronic Design. *GR-CPCI-XC2V*. [Accessed 7-May-2013]. URL: `http://www.pender.ch/products\_cpci\_xc2v.shtml`.

[74] S.N. Piramanayagam and T.C. Chong. *Developments in Data Storage: Materials Perspective*. Wiley, 2011.

[75] Moinuddin K. Qureshi. "Pay-As-You-Go: Low-Overhead Hard-Error Correction for Phase Change Memories". In: *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 '11*. New York, New York, USA: ACM Press, 2011, pp. 318 –328.

[76] Moinuddin K. Qureshi et al. "Enhancing Lifetime and Security of PCM-based Main Memory with Start-gap Wear Leveling". In: *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 42. New York, New York: ACM, 2009, pp. 14–23.

[77] Musfiq Rahman, Bruce Childers, and Sangyeun Cho. "StealthWorks: Emulating memory errors". In: *Proceedings of the First international conference on Runtime verification* (2010), pp. 360–367.

[78] P. Rech et al. "A Memory Fault Simulator for Radiation-Induced Effects in SRAMs". In: *2010 19th IEEE Asian Test Symposium*. IEEE, Dec. 2010, pp. 100–105.

[79] J. Rodgers et al. "A 4-Mb Non-volatile Chalcogenide Random Access Memory designed for space applications: Project status update". In: *Non-Volatile Memory Technology Symposium, 2008. NVMTS 2008. 9th Annual*. Nov. 2008, pp. 1–6.

[80] B. de Salvo. *Silicon Non-Volatile Memories: Paths of Innovation*. ISTE. Wiley, 2009.

[81] Stuart Schechter et al. "Use ECP, Not ECC, for Hard Failures in Resistive Memories". In: *Proceedings of the 37th Annual International Symposium on Computer Architecture*. ISCA '10. Saint-Malo, France: ACM, 2010, pp. 141–152.

[82] International Technology Roadmap for Semiconductors. *Process Integration, Devices, and Structures*. 2013. URL: `http://www.public.itrs.net/Links/2013ITRS/2013Chapters/2013PIDS.pdf` (visited on 05/21/2014).

[83] Nak Hee Seong, Dong Hyuk Woo, and H.-H.S. Lee. "Security Refresh: Protecting Phase-Change Memory against Malicious Wear Out". In: *Micro, IEEE* 31.1 (Jan. 2011), pp. 119–127.

[84] Nak Hee Seong et al. "SAFER: Stuck-At-Fault Error Recovery for Memories". In: *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*. Dec. 2010, pp. 115–124.

[85] A. Seznec. "A Phase Change Memory as a Secure Main Memory". In: *Computer Architecture Letters* 9.1 (Jan. 2010), pp. 5–8.

[86] A. Sheikholeslami and P.G. Gulak. "A survey of circuit innovations in ferroelectric random-access memories". In: *Proceedings of the IEEE* 88.5 (May 2000), pp. 667–689.

[87] P. Skoncej. "Fault Injection Framework for embedded memories". In: *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on*. Oct. 2013, pp. 77–82.

[88] P. Skoncej. "Low-Overhead Fault Injection Simulation and Emulation Technique for Embedded Memories". In: *Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*. Feb. 2014.

[89] P. Skoncej. "Single Error plus Single Erasure Correction with Redundancy Repair Scheme for Memory Reliability Improvement". In: *Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*. Feb. 2012.

[90] P. Skoncej. "WBR - word- and block-level hard error repair for memories". In: *Non-Volatile Memory Technology Symposium (NVMTS), 2012 12th Annual*. Oct. 2013, pp. 41–46.

[91] P. Skoncej and G. Schoof. "Getting ready for non-volatile memories". In: *2nd Workshop on Resilient Architectures*. Dec. 2011.

[92] P. Skoncej and G. Schoof. "Non-volatile memory controller design using fault-tolerant techniques for memory reliability improvement". In: *Smart Systems Integration*. Mar. 2011.

[93] C. Slayman. "Soft error trends and mitigation techniques in memory devices". In: *Reliability and Maintainability Symposium (RAMS), 2011 Proceedings - Annual*. Jan. 2011, pp. 1–5.

[94] C.H. Stapper and H.-S. Lee. "Synergistic fault-tolerance for memory chips". In: *Computers, IEEE Transactions on* 41.9 (Sept. 1992), pp. 1078–1087.

[95] Chin-Lung Su, Rei-Fu Huang, and Cheng-Wen Wu. "A processor-based built-in self-repair design for embedded memories". In: *Test Symposium, 2003. ATS 2003. 12th Asian*. Nov. 2003, pp. 366–371.

[96] Chin-Lung Su, Yi-Ting Yeh, and Cheng-Wen Wu. "An integrated ECC and redundancy repair scheme for memory reliability enhancement". In: *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on*. Oct. 2005, pp. 81–89.

[97] C.-E.W. Sundberg. "Erasure and Error Decoding for Semiconductor Memories". In: *Computers, IEEE Transactions on* C-27.8 (Aug. 1978), pp. 696–705.

[98] W.K.S. Walker, C.-E.W. Sundberg, and C.J. Black. "A Reliable Spaceborne Memory with a Single Error and Erasure Correction Scheme". In: *Computers, IEEE Transactions on* C-28.7 (July 1979), pp. 493–500.

[99] Fei Wang and Xiaolong Wu. "Non-volatile Memory Devices Based on Chalcogenide Materials". In: *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on*. Apr. 2009, pp. 5–9.

[100] Xueqiang Wang et al. "An on-chip high-speed 4-bit BCH decoder in MLC NOR flash memories". In: *Solid-State Circuits Conference, 2009. A-SSCC 2009. IEEE Asian*. Nov. 2009, pp. 229–232.

[101] M. White, Jin Qin, and J.B. Bernstein. "A study of scaling effects on DRAM reliability". In: *Reliability and Maintainability Symposium (RAMS), 2011 Proceedings - Annual*. Jan. 2011, pp. 1–6.

[102] B. Wongchaowart, M.K. Iskander, and Sangyeun Cho. "A content-aware block placement algorithm for reducing PRAM storage bit writes". In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. May 2010, pp. 1–11.

[103] Chi-Feng Wu, Chih-Tsun Huang, and Cheng-Wen Wu. "RAMSES: a fast memory fault simulator". In: *Proceedings 1999 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (EFT'99)*. IEEE Comput. Soc, 2002, pp. 165–173.

[104] Yuan Xie. "Modeling, Architecture, and Applications for Emerging Memory Technologies". In: *Design Test of Computers, IEEE* 28.1 (Jan. 2011), pp. 44–51.

[105] Doe Hyun Yoon and M. Erez. "Virtualized ECC: Flexible Reliability in Main Memory". In: *Micro, IEEE* 31.1 (Jan. 2011), pp. 11–19.

[106] Doe Hyun Yoon et al. "FREE-p: Protecting non-volatile memory against both hard and soft errors". In: *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*. Feb. 2011, pp. 466–477.

[107] K. Zhang. *Embedded Memories for Nano-Scale VLSIs*. Integrated Circuits and Systems. Springer, 2009.

[108] W.S. Zhao et al. "Embedded MRAM for high-speed computing". In: *VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on.* Oct. 2011, pp. 37–42.